2019211649

程序设计实践

实验报告

姓名:	吉诺
学号:	2019211649
报告日期:	December 16, 2021

Contents

1 系统设计 **2**
1.1 交互架构设计 . . . 2
1.1.1 后端交互架构设计 . . . 3
1.1.2 前端架构设计 . . . 5
1.1.3 安全性 . . . 5
1.2 高并发系统架构设计 . . . 6
1.3 客服机器人服务设计 . . . 8
1.3.1 DSL 定义 . . . 8
1.3.2 解析器 . . . 10
1.3.3 客服机器人模型 . . . 13
1.3.4 脚本加载器 . . . 15

2 程序实现 **15**
2.1 后端 . . . 15
2.1.1 站点配置 . . . 15
2.1.2 控制器 . . . 15
2.1.3 业务层数据接口 . . . 20
2.1.4 鉴权 . . . 21
2.1.5 业务层客服机器人 . . . 26
2.2 前端 . . . 26
2.2.1 UI 设计 . . . 26
2.2.2 `axios` 封装 . . . 28
2.2.3 `Portal` . . . 30
2.2.4 `Chatbox` . . . 32

3 接口文档 **35**
3.1 网络接口 . . . 35
3.1.1 `Sign` 接口 . . . 35
3.1.2 `Portal` 接口 . . . 35
3.1.3 `Chat` 接口 . . . 37

4 构建与部署 **38**
4.1 单元测试 . . . 38
4.2 容器化部署 . . . 44

5 自动化 **45**
5.1 CI/CD . . . 45

支持自定义的可编程客服机器人的设计与实现

2019211649 吉诺

(Dated: December 16, 2021)

任务要求：领域特定语言（Domain Specific Language，DSL）可以提供一种相对简单的文法，用于特定领域的业务流程定制。本作业要求定义一个领域特定脚本语言，这个语言能够描述在线客服机器人（机器人客服是目前提升客服效率的重要技术，在银行、通信和商务等领域的复杂信息系统中有广泛的应用）的自动应答逻辑，并设计实现一个解释器解释执行这个脚本，可以根据用户的不同输入，根据脚本的逻辑设计给出相应的应答。

1 系统设计

本系统基于前后端分离的交互模式，使用 Django 与 VueJS 框架开发。

1.1 交互架构设计

本节将展现本系统的数据流通与用户交互。

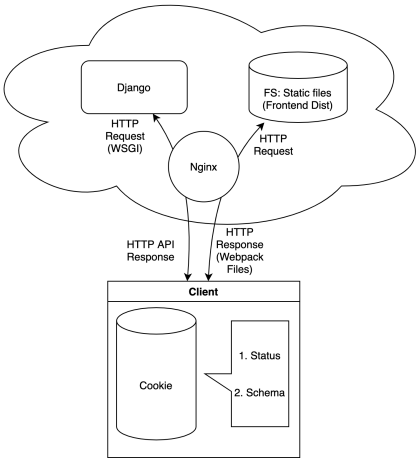

Figure 1: 基本架构图

工作流程 用一个三元组（`stage`，`schema`，`status`）描述客户端的状态。其中 `stage` 的含义是目前客户机的 UI 在哪一个窗口；`schema` 表示目前客户机选中哪一种客服机器人方案；`status` 表示目前客户机在对话中自动机所处的状态。这个元组存储在客户机本地，`stage` 存储在客户机的浏览器内存中；`schema` 与 `stage` 经过签名存储在客户机浏览器的站点 `Cookie` 中。

stage 可取值为 0,1，0 表示客户机目前处于 Portal 页面，1 表示客户机目前处于 Chatbox 页面。stage 的初始值为 0，页面刷新时，其值也会被设置为 0，也就是说，在用户刚进入页面时与刷新页面时，均会进入 Portal 页面。关于 Portal 与 Chatbox 的细节会在下文体现。

在 Portal 页面，客户机前端会向后段发起一个请求，查询可选 schema（方案）列表。得到结果之后，会将方案列表展示给用户，让用户选择方案。

当用户选中一个 schema（方案）后，前端会再次向后端发起请求，查询所选方案的设定信息，以加载自定义的客户机 Chatbox 文案，完成该操作后，会切换到 Chatbox 页面，同时 stage 被设置为值 1。

用户进入 Chatbox 页面后，前端会向后端发起初始化请求，请求内容包含所选方案名称，后端会为前端签发一个经过**签名**的 session_key，里面包含 status(=0) 与 schema 两个信息，客户端保存该信息，由于被后端使用私钥签名，前端不可修改，可视为锁定所选方案（会话期间不可修改），并将 status 初始化为 0。

这时，用户可以在 Chatbox 页面与客服机器人对话，每发送一条消息，前端均会向后端发起一次请求，内容为消息字符串。后端接收到请求后，会将用户消息输入构建好的自动机进行处理，将自动机的处理结果——即回复信息以及更新后的 status，返回给客户机前端。

当用户刷新页面，便认为用户退出本次对话，客户机前端不请求后端，清空浏览器内存，stage 自动恢复为 0，用户回到 Portal 页面。

工作流程图解如下。

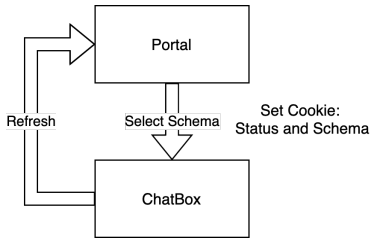

Figure 2: 前端工作流程

用户系统 基于 JWT 构建了用户系统，用户数据存储在后端。前端通过路由钩子实现了登录跳转功能，用户未登录时会自动跳转到登录页面，若登录但过期会在第一次网络通讯时提示用户重新登录。登录功能没有影响 Portal 的逻辑。

内容服务器与反向代理 Nginx 在本架构中提供两个功能，分别是静态内容服务器与反向代理。

由于使用前后端分离的交互模式，在生产环境，前端是 html、css、js 与其他静态资源文件，而后端是动态的网络接口，所以需要将前端与后端接口分离。首先，将 Nginx 配置为静态内容服务器，目录指向前端 WebPack 打包后的 dist 文件。然后，将 api/目录设置为反向代理地址，连接 Django 提供的 WSGI 接口，配置转发 $host（主机名），实现单域名前后端，并且避免跨站问题。

1.1.1 后端交互架构设计

后端架构主要涉及到两个概念，分别是**站点**与**应用**，这里的站点 Bank Service 表示搭载客服机器人服务的站点，其中包含了机器人服务的相关配置。这里的 Auth 与 Bot Service 分别表示鉴权服务

与机器人服务。两个服务的接口路径位于 `Bank Service` 下辖 `api/`路径下。

`Bot Service` 应用依赖于 `Auth` 应用，在 `Portal` 内的所有操作都需要登录才可以进行。

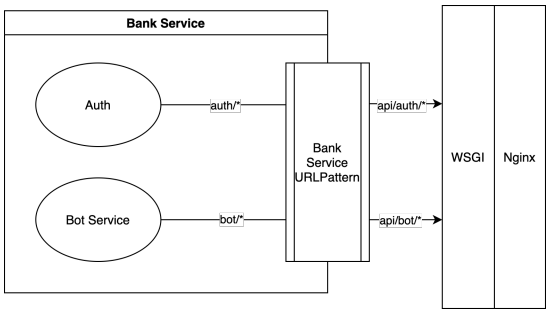

Figure 3: 后端基本架构

`Bot Service` 架构设计 后端在 MTC 设计模式的基础上进行设计。MTC 是 MVC 的修改版本，与 MVC 设计模式基本一致。MVC 模型全称 Model-View-Controller（模型-视图-控制器）三层模型，Django 内置后端渲染功能，用 Template（模板）替换了传统 MVC 中的视图，所以称之为 MTC 设计模式。使用 Django 开发前后端分离 WebApp 时，由于不再使用后端渲染，所以只保留 Model 与 Controller 两个模块。

Model 层在本项目中被命名为 Service（服务）层，抽象为一种服务，该服务不考虑网络接口，默认输入均为合法内容，完成业务逻辑的处理，不考虑 IO 控制。Controller 层负责完成接口的构建，不负责任何业务逻辑的处理，但需要维护一个用户友好的网络接口，能够识别非法输入，并且在业务层出现问题时给予用户合理的响应。Controller 在确认输入合法时，将输入封装交付给业务层，然后将业务层的处理结果封装返回给用户。

下图展示了本项目后端机器人 WebApp 的架构，用户输入经过网络到达控制器，控制器获得的请求体经过 `JSONSchema` 校验器验证后将请求交付给业务层的数据接口，数据接口调用更底层的**模型接口**获得处理结果。处理结果经过 Controller 的封装，通过网络返回给用户。（JSONSchema 与通讯格式会在下文描述）

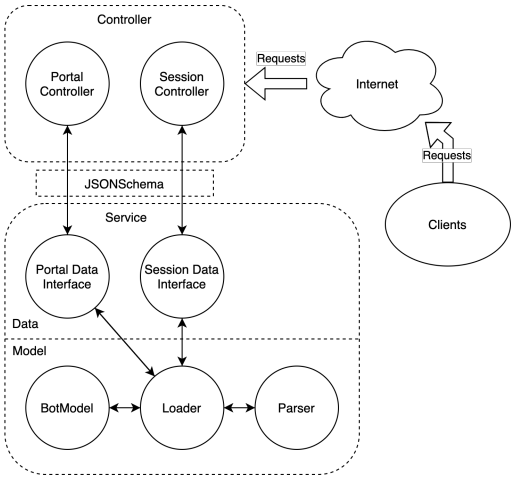

Figure 4: 后端基本架构

通讯与校验 对外开放的后端接口均使用 JSON 进行通讯，而在控制器将输入交付给业务层数据接口前，需要对用户发送的 JSON 进行校验，确保格式合法。为完成该功能，这里引入了 JSONSchema。

JSONSchema 本身是一个 JSON 对象，它用以描述另一个 JSON 的特征。在后端接口的设计中，一般使用 JSONSchema 来进行 JSON 数据格式验证，在数据提交到业务层次之前进行 JSON 格式的验证。使用 JSONSchema 进行 JSON 校验可以大幅提高后端开发效率。

1.1.2 前端架构设计

前端基于 `Vue3+TypeScript` 构建，使用 `Element Plus` 组件库快速成型。基本架构如下图。

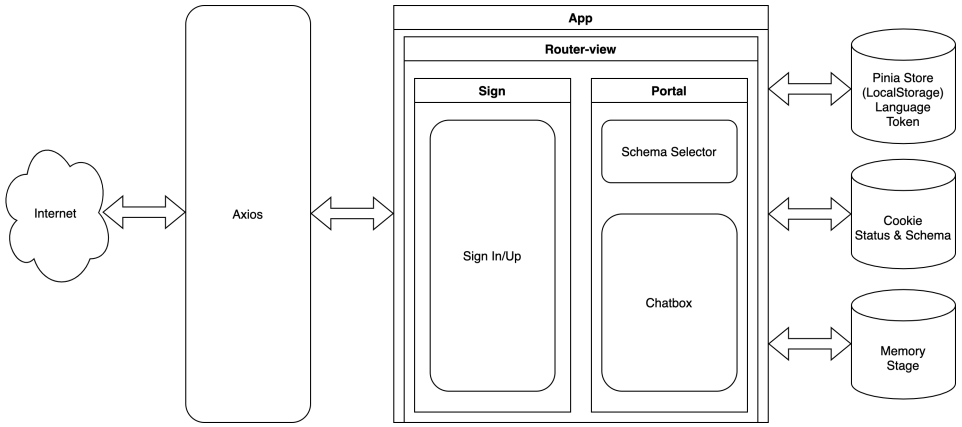

Figure 5: 前端基本架构

视图 根组件——App 下是 `Router-view` 组件，由该组件控制路由，`/`与`/sign` 分别为 `Portal` 与 `Sign` 页面。`Portal` 页面控制 `Schema Selector` 与 `Chatbox`。

存储 由于 `Vuex4`(Vue 生态中官方开发的状态管理库)对 `TypeScript` 的支持不佳，本项目使用 `Pinia` 进行状态管理，其中存储 `Token`，`Name` 与 `Language` 信息。状态机 `schema` 与 `status` 信息存储于 `Cookie` 中便于后端修改。内存中储存 `stage` 临时信息。

Axios 封装 客户端对网络请求进行了抽象，使用 `Axios` 库进行 `XHR` 请求，并且对 `Axios` 库进行了封装。在发起请求时，`Axios` 会在请求头中添加 `Authorization` 字段，内容为本地保存的 `JSON Web Token`；得到响应后，会更新本地 `Token` 为服务器返回的 `Authorization` 字段值。

i18n——多国语言支持 客户端添加了多国语言支持。前端可以自动识别用户的系统语言，将界面文案更换为相应的语言。

1.1.3 安全性

安全性是衡量一个 `WebApp` 优劣的重要因素。本系统的后端已经完全无状态化，所有状态信息均存储于前端用户侧，保护系统的安全是非常重要的。`schema` 与 `status` 分别是用户选择的方案与自动机当前的状态，若这两个字段可以被用户任意修改，可能会导致后端自动机出现访问越界的情况，产生安

全隐患。所以这里对 `Cookie` 进行了签名，仅后端可以生成签名，使 `schema` 与 `status` 在前端相当于只读。`JWT` 存储在 `localStorage` 不容易被 `CORS` 攻击劫持，并且本系统无敏感数据，即使拥有 `token` 也无法对系统造成危害。与此同时，用户的密码在发送前经过了 `md5` 加盐哈希处理，数据库泄露对用户隐私不造成危害。

1.2 高并发系统架构设计

本 WebApp 的设计过程中应用了大量的微服务设计方法，已有的基础上稍加改动，可以设计出支持高并发的 Web 客服机器人系统。

集群多实例 本系统的机器人客服服务后端不在内存保存任何用户数据（仅 `Auth` 使用数据库），所以可以直接使用 `Kubernetes` 将后端多实例化，配合 `k8s` 提供的 `Load Balancer`（负载均衡器）可以轻松提高系统的并发能力（Python 单进程对多核心 CPU 的利用能力很差，多实例化可以极大提升性能）。

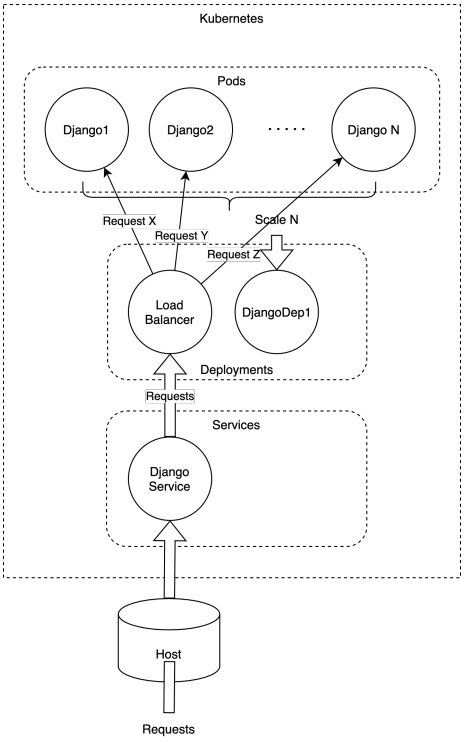

Figure 6: k8s 集群设计

缺点——较大的内存消耗 一个客服机器人服务需要能够提供多个方案的服务。换言之，每一个客服机器人后端实例都需要维护多个自动机副本，假设有 100 个方案储存在服务器上，假设 k8s 开启了 50 个客服机器人实例，那么就会有 5000 个自动机保存在内存中，并且 100 个方案每个方案都有 50 个一模一样的自动机。

享元模式 享元模式（Flyweight Pattern）主要用于减少创建对象的数量，以减少内存占用和提高性能。这种类型的设计模式属于结构型模式，它提供了减少对象数量从而改善应用所需的对象结构的方式。享

元模式尝试重用现有的同类对象，如果未找到匹配的对象，则创建新对象。

优化——时间、空间的平衡 虽然容器实例之间无法共享内存，但我们可以考虑享元模式的特点。享元模式的本质是为系统提供了一个高速缓存，不同的对象可以在上面共享内容。借鉴享元模式的思路，本系统也可以引入一个高速缓存，用以保存状态机。

内存数据库可以作为一个极佳选择，引入 Redis 以保存状态机。经过性能测试，Redis 的QPS在 50K 以上，假设一个客服机器人实例可以负担 200 的 QPS，那么一个 Redis 实例足以驱动 250 个客服机器人实例。

如果想支撑更大的 QPS，可以引入开启多主从集群模式的 Redis，每一个 Redis 实例保存同样的数据副本，在某个 Redis 认为自己的负载过高时，会将连接重定向到另一个 Redis 实例。如下图。

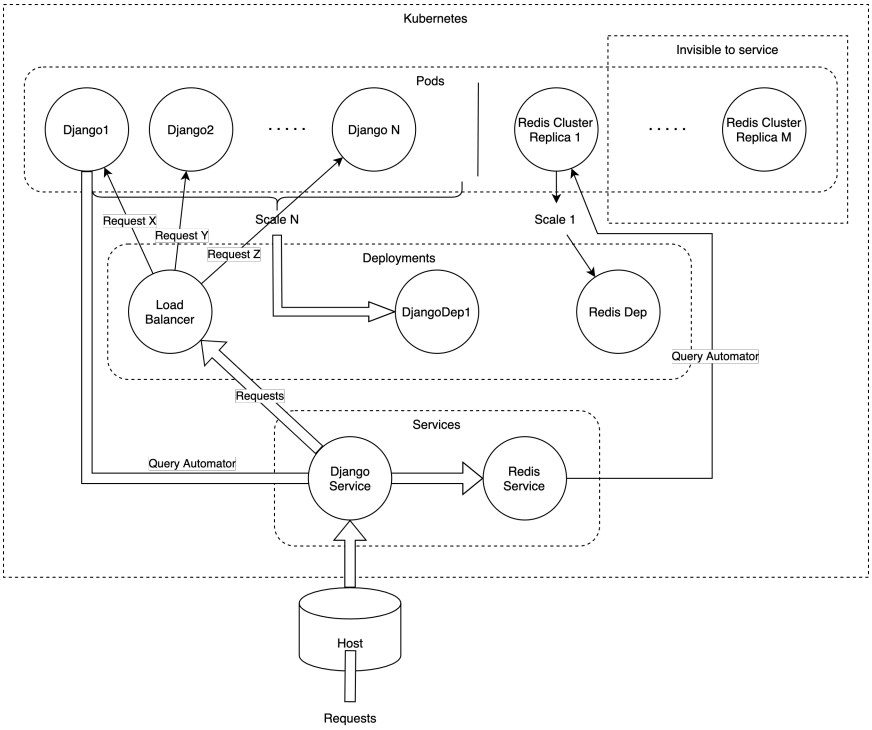

Figure 7: k8s 集群设计（优化）

可以发现，Deployments 只指向了一个 Redis 主实例。这是因为 Redis 的重定向由 Redis 自身控制，并且十分频繁，如果依然使用 Service 访问的话，会加大 Load Balancer 的负载，并且可能使重定向失效。

在实际使用中，Redis 重定向也是直接返回 k8s 容器网络的内网 IP，而非服务名称。

使用 Redis 保存自动机显然会增加请求处理的时间消耗，但是 Redis 实例越多时间越短，Redis 实例多到一定程度就不如每个实例都保存所有自动机，时间消耗也最短，内存消耗也最大。由此可见，使用 Redis 是在时间与空间的消耗间取得一个平衡。

1.3 客服机器人服务设计

客服机器人主要由三个模块组成，分别是 parser、bot 与 loader。其中，parser 负责将一个机器人定义脚本解析为脚本数据结构；bot 提供机器人消息响应的接口，并且可以通过脚本数据结构建立对应的机器人模型；loader 负责加载本地储存的脚本文件，并使用 parser 解析，然后将解析得到的脚本数据结构交给 bot 用以构建机器人模型，最后将机器人模型保存在内存中。

1.3.1 DSL 定义

这里先给出一个样例 DSL 文件，这个样例被部署在样例机器人。

```
settings
    "name": "柜台小二 甲零零壹"
    "title": "钱庄业务"
    "welcome": "以下是本庄可以提供的选项~"
    "subservice": "子服务: "
    "option": "服务选项: "
    "am": "午时"
    "pm": "午後"
    "cancel": "取消"
    "cancel_info": "输入“取消”可以取消操作～"
    "cancel_success": "取消成功～"
    "back": "返回"
    "back_info": "在子服务输入“返回”可以返回上一级服务～"
    "back_success": "返回成功～"
    "other": "如果有其它问题也可以直接和我讲~"
    "unknown": "祈蒙见恕！小二没理解少侠的意思..."
    "error": "未知错误"

service "理财"
    text "产品介绍" "A产品: 中风险...; B产品: 低风险; C产品: 高风险"
    script "理财产品推荐评估" "evaluate"
    faq "常见问题"
        "我适合什么样的理财产品? ": "选择理财产品需要根据自己对风险的承受能力，可以问我
            “理财产品推荐评估”进行评估。"
        "七日年化利率与年利率有什么区别? ": "七日年化利率是通过过去七日的收益情况估计的年
            利率，未来具有不确定性；而年利率是固定的，没有不确定性。"
        "如何使用定期投资功能? ": "前往首页，在金融功能栏下可以看到定期投资功能。"

service "基础金融业务"
    text "产品介绍" "手机银行APP可以使用转账、查询、账户管理、新卡申请、新卡激活、银行卡
        管理等功能"
```

```
29    script "附近网点查询" "输入你的位置" "querybranch"
```

本系统定义的机器人声明脚本拥有如下的语法特征：

- 使用缩进表示代码块，支持任意长度的空格缩进；
- 更长的缩进表示更低的级别，更低级别的代码块从属于上方的第一个更高级别语句；
- 一行只能有一个语句，即语句以换行符为句尾。

本系统定义的机器人声明脚本拥有以下语句：

根 `root`

文档的根，是隐藏的语句，所有缩进为 0 的语句均属于文档的根。

设置 `settings`

本语句只有一个 `settings` 关键字，表示其代码块为设定内容，本语句的代码块用于描述客服机器人的信息（如客服名称、业务名称），并且决定显示在客户端的文本内容。

`settings` 语句的代码块中仅允许出现**键值对**语句。

考虑到随着版本迭代设置选项可能会频繁修改，所以这里并没有对 settings 的键值对做严格的限定，所有选项均为可选选项（机器人模型内有默认值）。目前版本的可选选项如下：

- name: 机器人客服的名称；
- title: 客服业务的标题；
- welcome: 进入某个子服务或者刚刚进入对话页面时菜单的欢迎语；
- subservice: 菜单子服务文案；
- option: 菜单服务选项文案；
- am: 上午时间文案；
- pm: 下午时间文案；
- cancel: 取消脚本执行的口令；
- cancel_info: 取消脚本的提示文案；
- cancel_success: 取消成功文案；
- back: 返回上一级服务口令；
- back_info: 返回上一级服务的提示文案；
- back_success: 返回成功文案；
- other: 其它问题文案；
- unknown: 未识别的输入文案；
- error: 未知错误文案。

子服务 `service <name>`

描述子服务的代码块，`<name>` 为出现在菜单中的子服务名称。表示其代码块为子服务选项列表。

`service` 语句的代码块中仅允许出现**子服务**语句、**文本选项**语句、**脚本选项**语句、**接受参数的脚本选项**语句和**常见问题**语句。

文本选项 `text <name> <answer>`

表示一个简单的回答选项，`<name>` 为出现在菜单中的服务选项名称，`<answer>` 为用户选择该选项时的回复内容。

`text` 语句不允许拥有代码块。

脚本选项 `script <name> <module>`

表示一个脚本选项。`<name>` 为出现在菜单中的服务选项名称，`<module>` 为用户选择该选项时所执行的 `Python` 模块。系统要求模块内必须包含 `handle()` 函数。用户选择该选项时，系统会执行指定模块的 `handle()` 方法，并回复返回值。该方法用于实现复杂的查询功能。

`script` 语句不允许拥有代码块。

接受参数的脚本选项 `script <name> <tips> <module>`

表示一个接受参数的脚本选项。`<name>` 为出现在菜单中的服务选项名称，`<tips>` 表示用户选择该选项后回复的内容，用于指导用户输入参数，`<module>` 为用户选择该选项并且输入参数后所执行的 `Python` 模块。系统要求模块内必须包含 `handle(arg: string)` 函数。用户选择该选项并输入参数后，系统会将用户输入的参数作为 arg 执行指定模块的 `handle(arg: string)` 方法，并回复返回值。该方法用于实现复杂的查询功能。

`script` 语句不允许拥有代码块。

常见问题 `faq <name>`

表示一个常见问答选项。`<name>` 为出现在菜单中的服务选项名称。该语句表示其代码块为常见问答内容，本语句的代码块用于作为未被其他选项捕获的消息的缺省查询集。所有没有命中其他选项的消息会被检查是否被问答语句代码块中任意键包含，如果包含，会返回所有包含它的 `QA` 问答对作为回复。如果不包含，说明用户输入了模型无法处理的输入。

`faq` 语句的代码块中仅允许出现**键值对**语句。

键值对 `<key>: <value>`

表示一个键值对，`<key>` 和 `<value>` 可以为任何内容，其含义视情况而定。在作为 `settings` 的子语句时表示设置选项；在作为 `faq` 的子语句时表示 `QA` 问答选项。

`K-V` 语句不允许拥有代码块。

1.3.2 解析器

解析器的任务是解析一个脚本文件，并输出解析得到的脚本对象，在过程中如果发生错误，需要输出错误以提示用户。发生错误后需要进行错误恢复，继续进行解析，确保一次解析可以输出全部错误。

一次解析任务有两个阶段：

- 结构解析：跳过空行，将缩进解析为语句级别，构造代码块，使用广义表储存（每一个代码块都是一个数组，代码块内一行非空文本是一个元素，代码块内的更低级别代码块也是一个数组）。
- 语法-语义分析：分析结构解析得到的脚本结构，逐行分析文本，尝试将文本解析为语句。若语句解析失败，抛出错误并继续分析。语义分析是指，每个代码块从属于某一个语句，而语句决定了从属于它的代码块内允许出现的语句，分析器会检测出现的语句是否合法，如果不合法会抛出错误并继续分析。若一个无法识别的语句包含代码块，则允许其代码块包含任意语句，以继续分析。

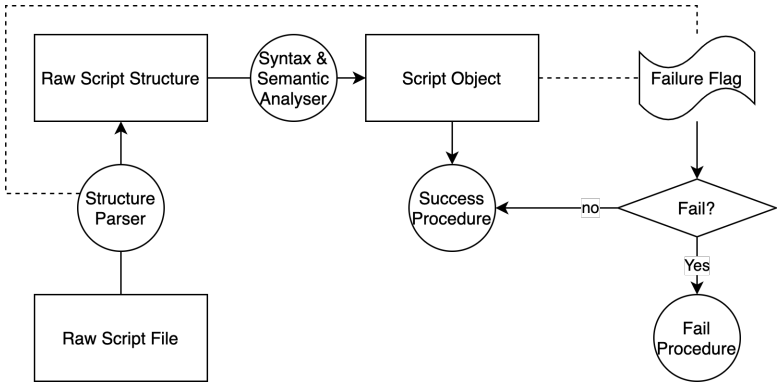

Figure 8: 解析器基本流程

为进行错误恢复，使用一个标识变量表示错误，分析结束后，如果存在错误标记，则解析失败，返回空。否则解析成功，返回得到的脚本对象。

结构解析器 结构解析器的工作流程如下：

1. 创建一个栈以存储代码块；
2. 逐行遍历脚本文件；
3. 若行文本为空（换行符、回车、空格），跳过；
4. 非空，获取缩进空格数目，储存为 `level`（值越高语句的从属级别越低），对行文本进行如下处理：
   - 若 `level` 与栈顶 `level` 相等，则说明它们属于同一代码块，将文本加入栈顶的代码块。
   - 若 `level` 大于栈顶 `level`，说明它属于栈顶元素的代码块，将本行语句与 `level` 压入栈中。并将本行文本构成的新代码块列表加入原栈顶元素的代码块中。
   - 若 `level` 小于栈顶 `level`，说明它属于栈顶元素代码块外侧的**某个**代码块，需要往前扫描栈寻找所属的代码块。扫描直到 `level` 与扫描到的代码块的 `level` 相等，将本位置之前的栈元素全部弹出，将本行文本加入新栈顶元素的代码块中。若扫描到栈元素的 `level` 小于行文本 `level` 时都没有找到相等的 `level`，说明发生了缩进错误，跳过该行并将错误标识设置为真。
5. 遍历结束，此时栈底的元素（0 级代码块）是结构化的脚本文本对象。取出该元素，作为语法-语义分析的输入。

语法-语义分析器 在说明语法-语义分析器的工作流程前，需要引入语句类型的概念。在本系统中，任何语句都属于以下几个类型：

- `Root` 根
- `Service` 子服务
- `Text` 文本
- `Script` 脚本
- `ScriptWaiting` 接受参数的脚本
- `FAQ` 常见问题
- `KVItem` 键值对
- `Setting` 设定
- `Any` 任意

除了 `Any` 类型，以上几个类型与 `DSL` 中定义的类型一一对应。`Any` 仅出现于语法-语义分析过程，表示一个语句是无法被识别的，此时这个语句的代码块内允许任意语句，这个类型用于错误恢复并继续分析。

语法-语义分析器使用了类似递归下降分析的方法，其分析函数（递归函数）接受两个参数，第一个是代码块父亲语句的类型，第二个是结构解析器得到的结构化的代码块，返回值为分析得到的语句列表，其流程如下：

1. 初始化一个语句列表 `script`；
2. 遍历输入的文本代码块列表；
3. 遍历时取当前元素 `elem` 的下一个元素 `next_elem`；
4. 若 `elem` 的类型是列表，跳过；
5. 使用语法分析工具识别该 `elem` 的文本，如果成功接受，则得到语句，否则输出语法错误信息，并将语句的类型设置为 `Any`。
6. 检查 `elem` 语句的类型是否在父语句的代码块中允许出现，如果不允许，输出非法语句错误。
7. 如果 `elem_next` 的类型是列表，说明 `elem_next` 是属于 `elem` 的代码块，执行分析函数的递归调用，参数为 `elem` 的类型与 `elem_next` 的代码列表。
8. 如果发生了上方的解析，则将解析结果（代码块语句列表）保存为列表 `block`，如果未发生上方的解析，则 `block` 为空。将 `elem` 解析得到的语句与 `block` 合成一个结构放入 `script` 中。

1.3.3 客服机器人模型

得到解析结果后，将脚本对象传递给客服机器人模块以建立机器人模型。客服机器人模型内有如下两个核心概念：

- 节点：根节点或服务节点，储存一个子服务的或者根的服务选项、子服务选项与 `FAQ` 信息。提供各类选项的 `getter` 和 `setter`。
- 状态转移表：客服机器人状态机的核心，每一个表项代表一个状态，每一列为不同条件的转移结果。状态分为以下三种：
  - `Root` 根：表示用户与机器人的会话处于根节点；
  - `Serv` 服务：表示用户与机器人的会话处于服务节点；
  - `Wait` 等待：表示用户与机器人的会话正在等待用户输入参数（带参数的脚本语句）。
- 转移函数：用户输入会进入转移函数，会根据是否满足转移条件与是否匹配节点中的选项决定下一个状态与回复的信息。

节点 节点用来储存根或者服务的服务选项，内有三个数据项，分别是 `query`、`faq_keyword` 与 `faq`。

- `query` 查询：储存 `text` 选项、`script` 选项与接受参数的 `script` 选项，三者注册的口令不允许重复，重复注册会抛出异常；
- `faq_keyword` `FAQ` 口令：用户输入 `FAQ` 口令时，节点会返回完整的 FAQ 文本，FAQ 口令不允许重复注册，重复注册会抛出异常；
- `faq` `FAQ` 集：当用户的输入未与上方两个数据项提供的口令匹配时，程序会在 `FAQ` 集中搜索用户输入，返回搜索结果。

节点数据结构本身不区分根与子服务，区分根与子服务是因为根没有上级服务，在状态转移表与转移函数中的处理与子服务不同。

状态转移表 这里基于DSL 定义的样例对应的状态转移表进行说明。主表：

status	type	node	prev	serv	token	wait	cancel
0	Root	0		S1			
1	Serv	1	0				
2	Serv	2	1			W1	
3	Wait	2			”附近网点查询”		2

子表 S1：

name	status
”理财”	1
”基础金融业务”	2

子表 W1:

name	status
”附近网点查询”	3

主表有四个项，表示有四个状态。`Root` 类型表示根节点，只有 `node` 与 `serv` 字段可有值；`Serv` 类型表示子服务，只有 `node`、`serv`、`prev` 和 `wait` 字段可有值；`Wait` 类型表示正在等待输入参数的接受参数的脚本选项，仅 `node`、`token` 与 `cancel` 字段可有值。

主表的 8 个字段分别表示：

1. status: 状态号；
2. type: 状态类型；
3. node: 所属节点；
4. prev: 返回时的上级状态；
5. serv: 下属子服务；
6. token: 选项名称；
7. wait: 下属接受参数的脚本选项；
8. cancel: 取消时的上级状态。

子表 `S1` 用以储存子服务状态转移关系。以第一个表项为例，当用户输入理财时，自动机的状态会转移到 `1`。

子表 `W1` 用以储存接受参数的脚本选项的状态转移关系。以第一个表项为例，当用户输入附近网点查询时，自动机会查询 `node[2]` 内的选项详细信息，输出一个参数提示，并转移到状态 `3`。

转移函数 用户的初始状态为 0。转移函数接受用户的输入 `msg` 与会话的当前状态 `status`，并且返回新的状态（也可能不变）与回复文本。下面是转移函数的工作流程：

- 若 `msg` 为空，`status` 为 0，返回进入根节点的欢迎信息；
- 若 `status` 的 `type` 为 `Wait`，进入以下流程：
  1. 如果 `msg` 为 `cancel` 口令，则取消执行接受参数的指令，根据转移表转移到上一级；
  2. 根据转移表的 `token` 在 `node` 中获取指令处理函数，调用该函数并返回结果。最后根据转移表转移到上一级。
- 若 `status` 的 `type` 为 `Serv` 或 `Root`，进入以下流程：
  1. 若 `status` 的 `type` 为 `Serv` 且 `msg` 为 `back` 口令，则表示退出本级子服务，根据转移表转移到上一级，并输出欢迎信息；
  2. 否则若 `msg` 在 `status` 的 `serv` 子表 `name` 字段中，则转移到对应的子服务状态，并输出欢迎信息；

3. 否则若 msg 在 status 的 wait 子表 name 字段中，则在 node 找到指令的详细信息，输出 tips，转移到对应的等待状态，等待用户输入参数；
4. 否则若 msg 在 status 的 node 的 query 集中，则返回对应的查询结果，不发生状态转移；
5. 否则若 msg 为 status 的 node 的 faq_keyword，则返回完整的 FAQ 文本，不发生状态转移；
6. 否则若 msg 在 status 的 node 的 faq 中搜素有返回值（可以搜到），则返回搜索到的 FAQ 文本，不发生状态转移；
7. 否则 msg 是无法识别的输入，返回 unknown 文本，不发生状态转移。

1.3.4 脚本加载器

脚本加载器负责将后端指定文件夹内的脚本全部加载到内存中，并建立对应的机器人模型，并且在后端运行过程中为业务层数据接口提供模型信息，并处理消息回复与状态转移。

脚本加载器在内存中维护一个字典，键为 schema 的标识符，值为 bot 机器人模型。

脚本加载器相当于连接后端业务层与机器人模型的桥梁。

2 程序实现

2.1 后端

2.1.1 站点配置

站点是 Django 内的概念。站点用于装载多个不同的应用，将他们分配到不同的路径下。

```
1  import bot_service.urls as bot_urls
2  import serv_auth.urls as auth_urls
3
4  from django.urls import path
5  from django.urls.conf import include
6
7
8  urlpatterns = [
9      path('api/bot/', include(bot_urls)),
10     path('api/auth/', include(auth_urls))
11 ]
```

bot_service 应用的 URL 被配置到 api/bot/下。

serv_auth 应用的 URL 被配置到 api/auth/下。

2.1.2 控制器

因为 serv_auth 提供的功能非常简单，没有使用分层设计，所以这里的控制器与业务层均指代 bot_service 的控制器与业务层。这里以 portal 接口的设计为例：

```
"""this module provides the interfaces of protal

Separeted controller module. It will handling the user requests
and make sure them verified. The verified requests will be packed
into dict then be sent to service layout. The benefit is extract
service operations from controllers easily and safely.

Typical usage example:
from django.urls import path

import portal


urlpatterns = [
    path('option', portal.option)
]
"""
import bot_service.service.data.portal as data
import bot_service.service.util.validate as validator
import serv_auth.auth as auth

from bot_service.service.util.resp import fail, success, expire

from django.http.request import HttpRequest
from django.http.response import JsonResponse


detail_schema = {
    'type': 'object',
    'required': ['schema'],
    'properites': {
        'schema': {'type': 'string'}
    }
}


@auth.preprocessToken
def option(request: HttpRequest) -> JsonResponse:
    """portal option api
```

```
40
41      Args:
42          request (HttpRequest): user request
43
44      Returns:
45          JsonResponse: response
46      """
47      try:
48          resp = data.option()
49      except Exception as e:
50          return JsonResponse(fail(str(e)))
51      return JsonResponse(success(resp))
52
53
54  @auth.preprocessToken
55  def detail(request: HttpRequest) -> JsonResponse:
56      """portal detail api
57
58      return the detail of asked schema
59
60      Args:
61          request (HttpRequest): user request
62
63      Returns:
64          JsonResponse: response
65      """
66      stat, res = validator.validate(request, detail_schema)
67      if stat == validator.FAIL:
68          return JsonResponse(res)
69
70      try:
71          resp = data.detail(res)
72      except Exception as e:
73          return JsonResponse(fail(str(e)))
74
75      return JsonResponse(success(resp))
```

程序中的文档注释按照 Google 的 `Python` 开发规范编写。

这个文件展示了网络接口的特征与 `JSONSchema` 的使用，网络接口中先通过 `JSONSchema` 校验请求体，然后再将格式化后的请求交给 `data`（业务层数据接口）处理。

校验器模块 后端基于 JSONSchema，简单封装了校验器，保存为 util.validate，校验函数实现如下：

```python
def validate(http_req: HttpRequest, schema: object) -> Tuple[bool, dict]:
    """validate a request with a schema

    Args:
        http_req (HttpRequest): an user http request
        schema (object): a json schema matches the wanted request

    Returns:
        Tuple[bool, dict]: (status, a verified request dict or failing response)
    """
    try:
        req = json.loads(http_req.body)
        jsonschema.validate(req, schema=schema)
    except JSONDecodeError:
        return FAIL, resp.fail("bad json format")
    except ValidationError as e:
        return FAIL, resp.fail(f"bad request body: {e.message}")
    return SUCCESS, req
```

该函数会校验请求，如果发现异常会返回包含错误信息的 JSON 响应（字典），否则，会返回一个请求体（字典）。

resp 封装 resp 模块封装了三种基本响应信息，分别是失败、成功与超时，控制器统一使用该封装生成响应，确保了响应体格式的统一。

```python
"""a simple module to generate specified response
"""
def fail(msg: str) -> dict:
    """when a request's response fails to make, use this

    Args:
        msg (str): message to describe details of the failure

    Returns:
        dict: the response
    """
    resp = {
        'code': 1,
        'msg': msg,
        'data': None
```

```
    }
    return resp


def expire(msg: str) -> dict:
    """DEPRECATED. when something like session expired, use this.

    Args:
        msg (str): message to describe details of the expiration

    Returns:
        dict: the response
    """
    resp = {
        'code': 2,
        'msg': msg,
        'data': None
    }
    return resp


def success(resp_body: dict | list=None) -> dict:
    """to generate the success response

    Args:
        resp_body (dict, optional): the data in response. can be a dict
        or list. Defaults to None.

    Returns:
        dict: the response
    """
    resp = {
        'code': 0,
        'msg': "success",
        'data': resp_body
    }
    return resp
```

2.1.3 业务层数据接口

数据接口接收控制器传递来的请求，处理完成后返回给控制器。数据接口要求传递来的请求是合法的。这里以 `portal` 数据接口为例。

```
"""this module provides the implementations of session api

Separeted service module. It will handling the verified user
requests passed from controllers. The verified request is a
legal dict object, which means the function inside this module
do not need any verification to see if the input is illegal.

Typical usage example:
try:
    resp = data.func(request)
except Exception as e:
    return JsonResponse(fail(str(e)))
return JsonResponse(success(resp))

"""
import bot_service.service.model.loader as loader

from django.contrib.sessions.backends.base import SessionBase


def option() -> list:
    """show all the options

    Args:
        session (SessionBase): a user session

    Returns:
        list: option list
    """

    rep = []
    for schema_id, bot in loader.bots.items():
        settings = bot.get_settings()
        title = settings['title']
        rep.append({
            'schema': schema_id,
```

```
            'title': title
        })
    return rep


def detail(req: dict) -> dict:
    """get detail of specified schema

    Args:
        session (SessionBase): a user session

    Returns:
        list: detail
    """
    bot = loader.bots.get(req['schema'])
    if bot is None:
        return {}
    return bot.get_settings()
```

2.1.4 鉴权

鉴权模块提供了装饰器 `preprocessToken`，在请求真正被 `API` 处理前，会被鉴权模块检查。鉴权模块会尝试对 `JWT` 进行解码，如果失败或超时，会回复携带对应错误码的响应。

鉴权模块还提供了登录/注册接口 `sign`，如果用户不存在会自动注册，存在则登录。

```
"""auth module. providing JWT preprocess and generation service
"""
from jwt import encode as encodeJWT, decode as decodeJWT

import functools
import io
import time
import jwt

import bot_service.service.util.validate as validator
import serv_auth.models as models

from bot_service.service.util.resp import fail, success, expire

from typing import Callable
```

```
16
17 from django.http.request import HttpRequest
18 from django.http.response import JsonResponse
19
20 from bank_service.settings import JWT_EXPIRE_IN, JWT_PUBLIC_PATH, JWT_SECRET_PATH
21
22
23 with io.open(JWT_SECRET_PATH, 'rb') as key:
24     JWT_SECRET = key.read() # load secret
25 with io.open(JWT_PUBLIC_PATH, 'rb') as pub:
26     JWT_PUBLIC = pub.read() # load public
27
28
29 class ExposeAuthorizationMiddleware:
30     """middleware to allow authorization read by frontend
31     """
32     def __init__(self, get_response):
33         self.get_response = get_response
34
35     def __call__(self, request):
36         response = self.get_response(request)
37         response['Access-Control-Expose-Headers'] = "Authorization"
38         return response
39
40
41 # 生成token
42 def generateToken(
43     userId: int = None,
44     name: str = None,
45     detail: dict = None,
46     payload: dict = None
47 ) -> str:
48     """generate a jwt
49
50     Args:
51         userId (int, optional): userid. Defaults to None.
52         name (str, optional): username. Defaults to None.
53         detail (dict, optional): detail in jwt rfc. Defaults to None.
54         payload (dict, optional): content to carry. Defaults to None.
```

```
55
56      Returns:
57          str: the token
58      """
59      if payload is not None:
60          token = encodeJWT(
61              payload,
62              JWT_SECRET, algorithm="RS256")
63      else:
64          token = encodeJWT({
65              'id': userId,
66              'name': name,
67              'exp': int(time.time()) + JWT_EXPIRE_IN,
68              'detail': detail
69          }, JWT_SECRET, "RS256")
70      if isinstance(token, bytes):
71          token = token.decode('utf-8')
72      return token
73
74
75  # 装饰器 更新token 检测过期并重定向
76  def preprocessToken(requestHandler: Callable) -> Callable:
77      """decorator. check token before access api
78
79      Args:
80          requestHandler (Callable): the api method
81
82      Returns:
83          Callable: warpped api method
84      """
85      @functools.wraps(requestHandler)
86      def wrapper(request: HttpRequest):
87          # update token here
88          token = request.META.get('HTTP_AUTHORIZATION')
89          if token is None:
90              return JsonResponse({
91                  'code': 50008,
92                  'msg': 'need login'
93              })
```

```
94          try:
95              try:
96                  payload = decodeJWT(token, JWT_PUBLIC, algorithms=['RS256'])
97                  payload['exp'] = int(time.time()) + JWT_EXPIRE_IN
98                  token = generateToken(payload=payload)
99              except jwt.ExpiredSignatureError:
100                 return JsonResponse({
101                     'code': 50014,
102                     'msg': 'token expired'
103                 })
104             except jwt.DecodeError:
105                 return JsonResponse({
106                     'code': 50008,
107                     'msg': 'token broken'
108                 })
109         except Exception as e:
110             print(e)
111             return JsonResponse({
112                 'code': 0,
113                 'message': 'error occured while handling token'
114             })
115         response: JsonResponse = requestHandler(request)
116         response['Authorization'] = token
117         return response
118     return wrapper
119
120
121 sign_schema = {
122     'type': 'object',
123     'required': ['username', 'pwd'],
124     'properties': {
125         'username': {'type': 'string', 'minLength': 4, 'maxLength': 32, 'pattern': r'[a-z0-9A
                -Z_]+'},
126         'pwd': {'type': 'string', 'length': 32}
127     }
128 }
129
130
131 def sign(request: HttpRequest):
```

```
132     """simple sign api
133
134     Args:
135         request (HttpRequest): user request
136
137     Returns:
138         JsonResponse: response
139     """
140     stat, res = validator.validate(request, sign_schema)
141     if stat == validator.FAIL:
142         return JsonResponse(res)
143
144     res['username'].lower()
145
146     reg = False
147
148     try:
149         user = models.User.objects.get(name=res['username'])
150     except models.User.DoesNotExist:
151         reg = True
152
153     if reg:
154         user = models.User()
155         user.name = res['username']
156         user.pwd = res['pwd']
157         user.save()
158
159         token = generateToken(user.id, res['username'])
160     else:
161         user: models.User
162         if user.pwd != res['pwd']:
163             return JsonResponse(fail('wrong password.'))
164         token = generateToken(user.id, res['username'])
165
166     resp = JsonResponse(success())
167     resp['Authorization'] = token
168     return resp
```

2.1.5 业务层客服机器人

本章节的代码篇幅过长，所以不再展示。

解析器 解析器完全按照系统设计章节提出的架构实现。

机器人模型 机器人模型完全按照系统设计章节提出的架构实现。

加载器 加载器完全按照系统设计章节提出的架构实现。

2.2 前端

主要展示 `Portal` 与 `Chatbox`。

2.2.1 UI 设计

前端实现了 `i18n` 自动语言切换功能，由于开发软件时使用的系统语言为英语，这里 UI 的语言显示为英语。

响应式 响应式用于实现对移动端的兼容，本系统的前端使用 `CSS` 的 `@media` 标签实现响应式。以屏宽 `960px` 为分界线，小于阈值的屏幕中聊天窗口切换至全屏，而大于这个阈值的屏幕则会显示固定大小的聊天窗口。

配色 最初设计中，聊天系统是模拟一个古代钱庄的前台服务，所以采用了传统的中国色配色。

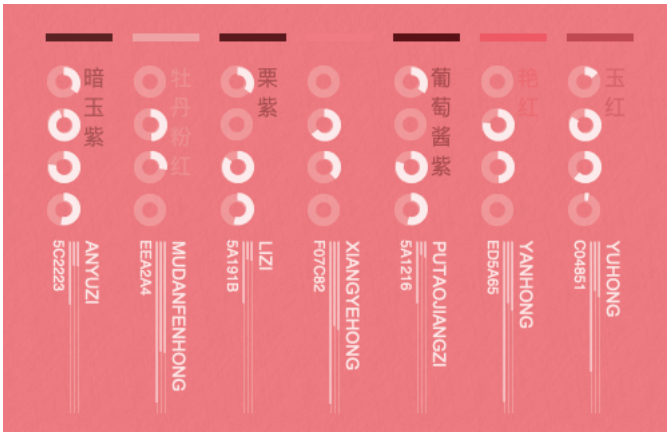

Figure 9: 配色

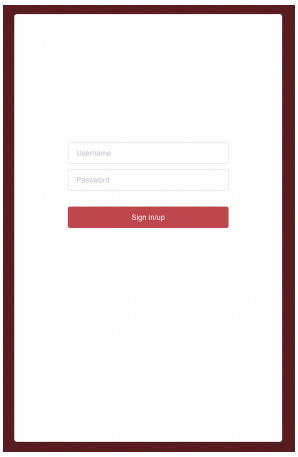

Figure 10: PC 端 Sign

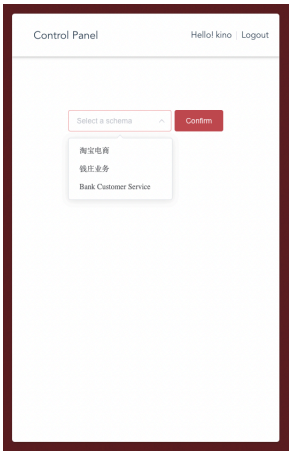

Figure 11: PC 端 Portal

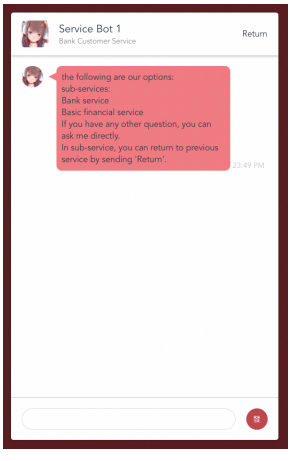

Figure 12: PC 端 Chatbox

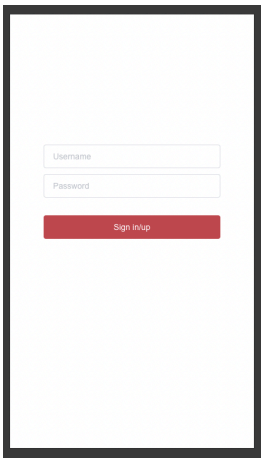

Figure 13: Mobile 端 Sign

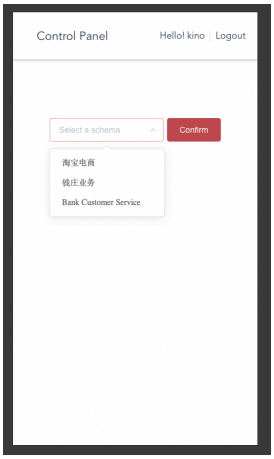

Figure 14: Mobile 端 Portal

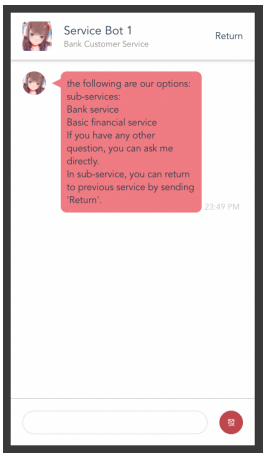

Figure 15: Mobile 端 Chatbox

2.2.2 axios 封装

```
import axios from 'axios'
import { ElMessageBox, ElMessage } from 'element-plus'
import { getToken, setToken } from '@/utils/auth'
import { useStore } from '@/store'
import i18n from '@/languages'

// create an axios instance
const service = axios.create({
  baseURL: process.env.VUE_APP_API, // url = base url + request url
  // withCredentials: true, // send cookies when cross-domain requests
  timeout: 5000 // request timeout
})

// request interceptor
service.interceptors.request.use(
  config => {
    // do something before request is sent
    const token = getToken()

    if (token.length > 0) {
      // let each request carry token
      // ['X-Token'] is a custom headers key
      // please modify it according to the actual situation
      // axios.defaults.headers.common['Authorization'] = getToken()

      // console.log(axios.defaults.headers.common['Authorization'])
      const header = config.headers as any
      header.Authorization = getToken();
    }
    return config
  },
  error => {
    // do something with request error
    console.log(error) // for debug
    return Promise.reject(error)
  }
)

```

```
39 // response interceptor
40 service.interceptors.response.use(
41   /**
42    * If you want to get http information such as headers or status
43    * Please return  response => response
44   */
45
46   /**
47    * Determine the request status by custom code
48    * Here is just an example
49    * You can also judge the status by HTTP Status Code
50    */
51   response => {
52     const res = response.data
53
54     // if the custom code is not 20000, it is judged as an error.
55     if (res.code !== 0 && res.code !== 1) {
56       ElMessage({
57         message: res.msg || 'Error',
58         type: 'error',
59         duration: 5 * 1000
60       })
61
62       // 50008: Illegal token; 50012: Other clients logged in; 50014: Token expired;
63       if (res.code === 50008 || res.code === 50012 || res.code === 50014) {
64         // to re-login
65         const t = i18n.global.t
66         ElMessageBox.confirm(t('message.relogin_text'), t('message.logout'), {
67           confirmButtonText: t('message.relogin'),
68           cancelButtonText: t('message.cancel'),
69           type: 'warning'
70         }).then(() => {
71           setToken('')
72           location.reload()
73         })
74       }
75       return Promise.reject(new Error(res.msg || 'Error'))
76     } else if (res.code == 0) {
77       setToken(response.headers['authorization'])
```

```
78        return res
79      } else {
80        return res
81      }
82    },
83    error => {
84      console.log('err' + error) // for debug
85      ElMessage({
86        message: error.message,
87        type: 'error',
88        duration: 5 * 1000
89      })
90      return Promise.reject(error)
91    }
92  )
93
94  export default service
```

axios 封装完成了登录过期校验，自动重定向等功能。Token 损坏或过期时会弹出如下窗口提醒用户重新登录。

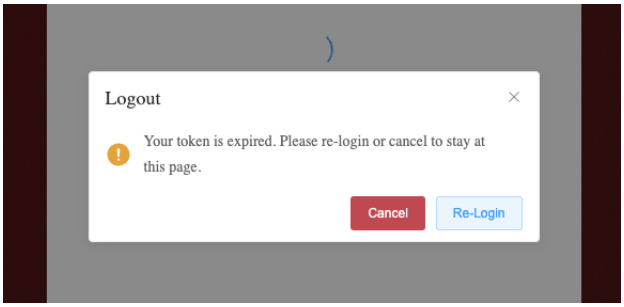

Figure 16: 注销提示

2.2.3 Portal

```
1  import Chatbox from "@/components/Chatbox.vue";
2  import { fetchOptions, fetchDetail } from "@/api/portal";
3  import { onBeforeMount, ref } from "vue";
4  import { openToast } from "toast-ts";
5  import { useI18n } from 'vue-i18n'
6  import { useStore } from '@/store'
7  import { setToken } from '@/utils/auth';
8
```

```
export default {
  name: "Portal",
  components: {
    Chatbox,
  },
  setup() {
    const stage = ref(0);
    const loading = ref(false);
    const schema = ref("");
    const boxStyle = ref({
    })
    const options: Array<{ schema: string; title: string }> = [];
    const am = ref("AM");
    const pm = ref("PM");
    const name = ref("bot1");
    const title = ref("service");
    const store = useStore()
    const username = store.$state.name


    const { t } = useI18n()

    onBeforeMount(async () => {
      loading.value = true;
      const options_ = await fetchOptions();
      options_.forEach((option) => {
        options.push({
          schema: option.schema,
          title: option.title,
        });
      });
      loading.value = false;
    });

    async function submit(): Promise<void> {
      if (schema.value === "") {
        openToast(t('message.sel_schema'));
        return;
      }
```

```
48      loading.value = true;
49
50      const detail = await fetchDetail({schema: schema.value})
51      name.value = detail.name
52      title.value = detail.title
53      am.value = detail.am
54      pm.value = detail.pm
55
56      stage.value = 1
57      loading.value = false
58    }
59
60    function logout() {
61      setToken('')
62      location.reload()
63    }
64
65    return {
66      stage,
67      loading,
68      schema,
69      options,
70      boxStyle,
71      name,
72      title,
73      am,
74      pm,
75      username,
76      submit,
77      logout
78    };
79  },
80 };
```

2.2.4 Chatbox

```
1 // 初始化函数
2 async mounted() {
3   let fail = false;
```

```
  let resp;
  try {
    resp = await fetchInit({ schema: this.schema });
    if (resp.code != 0) {
      fail = true;
    }
  } catch (error) {
    console.error(error);
    fail = true;
  }

  if (resp === undefined) {
    openToast(this.t('message.error'));
    return;
  }

  if (fail) {
    this.disabled = true;
    this.message = this.t('message.service_err');
    return;
  }

  let msgList = resp.data;

  msgList.forEach((msg) => {
    this.loadMsg(true, msg.content, msg.time);
  });
}

// 加载并回显消息
async loadMsg(bot: boolean, msg: string, time: number) {
  this.history.push({
    msg_id: this.maxMsgId++,
    bot: bot,
    content: msg,
    time: time,
  });
  await this.$nextTick();
  let historyContent = this.$refs.historyContent as any;
```

```
43    let historyScrollbar = this.$refs.historyScrollbar as any;
44    historyScrollbar.setScrollTop(historyContent.clientHeight);
45  }
46
47  // 发消息函数
48  async send() {
49    let msg: string = this.message;
50    if (msg.length == 0) {
51      return;
52    }
53    let time: number = new Date().getTime();
54    this.message = "";
55    this.loadMsg(false, msg, time);
56
57    try {
58      let resp = await fetchMessage({
59        content: msg,
60      });
61
62      if (resp.code == 2) {
63        msg = this.t('message.expire');
64        time = new Date().getTime();
65        this.loadMsg(true, msg, time);
66      } else if (resp.code != 0) {
67        msg = this.error;
68        time = new Date().getTime();
69        this.loadMsg(true, msg, time);
70      } else {
71        let data = resp.data;
72        data.forEach((elem) => {
73          this.loadMsg(true, elem.content, elem.time);
74        });
75      }
76    } catch (error) {
77      console.error(error);
78      msg = this.error;
79      time = new Date().getTime();
80      this.loadMsg(true, msg, time);
81    }
```

```
}
```

3 接口文档

3.1 网络接口

样例来自样例机器人

3.1.1 Sign 接口

/auth/sign POST

功能：登录/注册。请求体：

```
{
    "username": "kino",
    "pwd": "<md5 hash string>"
}
```

响应头：

```
...
authorization: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.
    eyJpZCI6MSwibmFtZSI6Imtpbm8iLCJleHAiOjE2Mzk1OTI1NTMsImRldGFpbCI6bnVsbH0.
    zYoOaElK4wKbKiADwUkXfabzRc0oi1YIb6-ADtUxqSnEe1wpaUCs2J2cwc9b0cSl-
    wey0B5a0fthD04qIq4Upq88IlcJhLdWQbcXURTeFM5t2g66ux6KtbL-
    qpo6nEt7PZfVkwTrw3ukCquO5Ch2n1jj-
    mnAkaONrGugq5QI6k612glcn_RZ9Tc8gg8ztpacMHJyLofYq5VuOXkXfENKAbRYsF22oz-4
    ZS5qgk8o_8TK2-eXWyLKVrVDeihZMpBIjJepTKDv83gKrtaWcJjWWQlze5OirWDLXUF1FJsDWqFQ-5
    a2fJF_AyGxjfCP0ltHw3of99H6je5No33YCPlnzw
...
```

响应体：

```
{
    "code": 0,
    "msg": "success",
    "data": null
}
```

3.1.2 Portal 接口

/portal/option GET

功能：列出可选择的方案。

请求体：无

响应体：

```
{
    "code": 0,
    "msg": "success",
    "data": [
        {
            "schema": "0",
            "title": "淘宝电商"
        },
        {
            "schema": "1",
            "title": "钱庄业务"
        },
        {
            "schema": "2",
            "title": "Bank Customer Service"
        }
    ]
}
```

/portal/detail POST

功能：获取指定方案的设定。

请求体：

```
{
    "schema": "0"
}
```

响应体：

```
{
    "code": 0,
    "msg": "success",
    "data": {
        "name": "电子客服",
        "title": "淘宝电商",
        "welcome": "以下是本店可以提供的服务",
        "subservice": "子服务：",
        "option": "服务选项：",
        "am": "上午",
```

```
        "pm": "下午",
        "cancel": "取消",
        "cancel_info": "输入“取消”可以取消操作～",
        "cancel_success": "取消成功～",
        "back": "返回",
        "back_info": "输入“返回”可以返回上一级服务～",
        "back_success": "返回成功～",
        "other": "如果有其它问题也可以直接和我讲~",
        "unknown": "抱歉！未能理解。",
        "error": "未知错误"
    }
}
```

3.1.3 Chat 接口

/chat/init POST

功能：初始化一个会话，设置 cookie 保存 schema 与 status。

请求体：

```
{
    "schema": "0"
}
```

响应头：

设置经过签名的 cookie。

```
...
Set-Cookie: sessionid=eyJzY2hlbWEiOiIwIiwic3RhdHVzIjowfQ:1mwjoH:yLLKGZaCeffWGsT-Mq1C3
    xTB8hcJp2jHsCCjdDaIeHk; expires=Mon, 13 Dec 2021 13:48:01 GMT; HttpOnly; Max-Age
    =7200; Path=/; SameSite=Lax
...
```

响应体：

```
{
    "code": 0,
    "msg": "success",
    "data": [
        {
            "content": "以下是本店可以提供的服务\n子服务: \n商品业务\n如果有其它问题也可
                以直接和我讲~\n输入“返回”可以返回上一级服务～",
            "time": 1639396081236
        }
```

```
    ]
}
```

/chat/message POST

功能：处理一条消息，更新 cookie 保存新的 status。

请求体：

```
{
    "content": "商品业务"
}
```

响应头：

设置经过签名的 cookie。

```
...
Set-Cookie: sessionid=eyJzY2hlbWEiOiIwIiwic3RhdHVzIjoxfQ:1mwk7n:Tp5h7Y-yinl6D7tuQMVxG
    5JnMDXg_T5OaD4CE1d37gE;
...
```

响应体：

```
{
    "code": 0,
    "msg": "success",
    "data": [
        {
            "content": "以下是本店可以提供的服务\n服务选项: \n产品介绍\n使用方法\n查价格\
                n如果有其它问题也可以直接和我讲~",
            "time": 1639397291725
        }
    ]
}
```

4 构建与部署

4.1 单元测试

本系统的后端拥有完善的单元测试，使用 Django 提供的反射策略执行单元测试。Django 提供了一键执行所有单元测试的功能，Django 会在已安装的 Application 中搜寻实现了 unittest 基类的类，并执行其中以 test 开头的方法。

此处展示用于测试解析器的 test_parser 模块。

```python
import logging
import unittest

import bot_service.service.model.parser as parser

from bot_service.service.model.bot import CommandEnum


class TestParser(unittest.TestCase):
    class __Capture:
        def __init__(self) -> None:
            self.__buffer = []

        def write(self, s: str) -> None:
            self.__buffer.append(s)

        def getline(self) -> str:
            return self.__buffer.pop(0)

        def getall(self) -> list[str]:
            buffer = self.__buffer
            self.__buffer = []
            return buffer

    def test_complicated(self) -> None:
        logger = logging.getLogger()
        logger.disabled = True
        with open('tests/testcase/complicated.def', 'r', encoding='utf8') as f:
            script = parser.load_script(f)

            assert script is not None
            assert script[0][0][0] == CommandEnum.Setting

            settings = script[0][1]
            assert len(settings) == 13
            assert settings[0] == ((CommandEnum.KVItem, 'name', 'name'), None)
            assert settings[12] == ((CommandEnum.KVItem, 'unkown', 'unkown'), None)

            assert script[1][0] == (CommandEnum.Service, 'service1')
```

```
40
41          services = script[1][1]
42          assert len(services) == 6
43          assert services[0] == ((CommandEnum.Text, 'text test1', 'text1'), None)
44          assert services[1] == ((CommandEnum.Script, 'script test1', 'script1'), None)
45
46          assert services[4][0] == (CommandEnum.FAQ, 'faq1')
47
48          faq = services[4][1]
49          assert len(faq) == 8
50          assert faq[0] == ((CommandEnum.KVItem, 'faq test1', 'answer1'), None)
51          assert faq[7] == ((CommandEnum.KVItem, 'faq test8', 'answer8'), None)
52
53          assert script[2][0] == (CommandEnum.Service, 'service2')
54
55          services = script[2][1]
56          assert len(services) == 6
57          assert services[0] == ((CommandEnum.Text, 'text test4', 'test4'), None)
58          assert services[2] == ((CommandEnum.Text, 'text test6', 'test6'), None)
59          assert services[4] == ((CommandEnum.ScriptWaiting, 'script test2', 'holder', '
                script2'), None)
60          assert services[5] == ((CommandEnum.Script, 'script test3', 'script3'), None)
61
62          assert services[3][0] == (CommandEnum.Service, 'service3')
63
64          services = services[3][1]
65          assert len(services) == 2
66          assert services[0] == ((CommandEnum.Text, 'text test7', 'test7'), None)
67
68          assert services[1][0] == (CommandEnum.FAQ, 'faq2')
69
70          faq = services[1][1]
71          assert len(faq) == 3
72          assert faq[0] == ((CommandEnum.KVItem, 'faq test9', 'answer9'), None)
73          assert faq[2] == ((CommandEnum.KVItem, 'faq test11', 'answer11'), None)
74
75      def test_empty(self) -> None:
76          logger = logging.getLogger()
77          logger.disabled = True
```

```
78          with open('tests/testcase/empty.def', 'r', encoding='utf8') as f:
79              script = parser.load_script(f)
80
81              assert script is not None
82              assert isinstance(script, list)
83              assert len(script) == 0
84
85      def test_incorrect(self) -> None:
86          logger = logging.getLogger()
87          logger.handlers.clear() # remove all the handlers
88          logger.filters.clear() # remove all the filters
89          logger.propagate = False # do not pass msg to the father logger
90          # add a handler for test
91          handler = logging.StreamHandler()
92          capture = TestParser.__Capture()
93          handler.setStream(capture)
94          logger.addHandler(handler)
95          logger.disabled = False # enable logger
96          with open('tests/testcase/incorrect.def', 'r', encoding='utf8') as f:
97              parser.load_script(f)
98              output = [s.strip() for s in capture.getall()]
99              assert 'bad indentation at line 8.' in output
100             assert 'sub-command not allowed at line 1.' in output
101             assert 'syntax error at line 3.' in output
102             assert 'sub-command not allowed at line 3.' in output
103             assert 'no sub definition allowed under command at line 12.' in output
104
105     def test_naughty_indent(self):
106         logger = logging.getLogger()
107         logger.disabled = True
108         with open('tests/testcase/naughty_indent.def', 'r', encoding='utf8') as f:
109             script = parser.load_script(f)
110
111             assert script is not None
112
113             assert script[0][0] == (CommandEnum.Service, 'service1')
114             assert script[1][0] == (CommandEnum.Service, 'service2')
115
116             services = script[0][1]
```

```
117             assert len(services) == 3
118             assert services[0] == ((CommandEnum.Text, 'text test1', 'text1'), None)
119             assert services[1] == ((CommandEnum.Script, 'script test1', 'script1'), None)
120             assert services[2][0] == (CommandEnum.FAQ, 'faq1')
121
122             faq = services[2][1]
123             assert len(faq) == 3
124             assert faq[0] == ((CommandEnum.KVItem, 'faq test1', 'answer1'), None)
125             assert faq[2] == ((CommandEnum.KVItem, 'faq test3', 'answer3'), None)
126
127             services = script[1][1]
128             assert len(services) == 2
129             assert services[0] == ((CommandEnum.Text, 'text test4', 'text4'), None)
130             assert services[1] == ((CommandEnum.Script, 'script test2', 'script2'), None)
131
132     def test_recursive(self):
133         logger = logging.getLogger()
134         logger.disabled = True
135         with open('tests/testcase/recursive.def', 'r', encoding='utf8') as f:
136             script = parser.load_script(f)
137
138             assert script is not None
139
140             assert script[0][0] == (CommandEnum.Service, '1')
141
142             services = script[0][1]
143             assert len(services) == 1
144             assert services[0][0] == (CommandEnum.Service, '2')
145
146             services = services[0][1]
147             assert len(services) == 1
148             assert services[0][0] == (CommandEnum.Service, '3')
149
150             services = services[0][1]
151             assert len(services) == 1
152             assert services[0][0] == (CommandEnum.Service, '4')
153
154             services = services[0][1]
155             assert len(services) == 1
```

```
156             assert services[0][0] == (CommandEnum.Service, '5')
```

该模块会进行五个用例测试，进行**复杂脚本解析**测试、**空脚本解析**测试、**错误脚本解析**测试、**任意缩进脚本解析**测试与**递归脚本解析**测试。

测试的方法是读取脚本，然后使用 `parser` 进行解析，解析完成后得到脚本对象，取出一些位置的脚本进行断言，如果全部通过，则测试成功。

错误脚本解析测试需要读取解析过程中的输出内容，而输出内容是使用日志模块打印在控制台的。所以这里将日志重定向到 `Capture` 对象，并且去除日志的所有格式与附加信息，然后在日志上进行断言，确保能够识别出文件内的全部错误。

下方展示两个测试用例的内容，依次是灵活缩进脚本与错误脚本：

```
service "service1"
 text "text test1" "text1"
 script "script test1" "script1"
 faq "faq1"
              "faq test1": "answer1"
              "faq test2": "answer2"
              "faq test3": "answer3"

service "service2"
                          text "text test4" "text4"
                          script "script test2" "script2"
```

错误脚本：

```
faq "root_faq"
service "理财"
    text "产品介绍" a "A产品：中风险...; B产品：低风险; C产品：高风险"
    script "理财产品推荐评估" "evaluate"
    faq "常见问题"
        "我适合什么样的理财产品？": "选择理财产品需要根据自己对风险的承受能力，可以问我“理财产品推荐评估”进行评估。"
        "七日年化利率与年利率有什么区别？": "七日年化利率是通过过去七日的收益情况估计的年利率，未来具有不确定性；而年利率是固定的，没有不确定性。"
       "如何使用定期投资功能？": "前往首页，在金融功能栏下可以看到定期投资功能。"

service "基础金融业务"
  text "产品介绍" "手机银行APP可以使用转账、查询、账户管理、新卡申请、新卡激活、银行卡管理等功能"
    script "附近网点查询" "querybranch"
```

执行测试，只需要在项目根目录下执行 `python manage.py test` 即可。测试成功时的结果如下：

```
System check identified no issues (0 silenced).
.....
----------------------------------------------------------------------
Ran 5 tests in 0.018s

OK
```

测试失败（这里将复杂脚本用例修改为错误的，导致第一个测试失败）时会输出如下结果:

```
System check identified no issues (0 silenced).
F....
======================================================================
FAIL: test_complicated (tests.test_parser.TestParser)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/kino/Documents/homework/program_designing/lab.dsl-chatbot/bank_service
      /tests/test_parser.py", line 31, in test_complicated
    assert script is not None
AssertionError

----------------------------------------------------------------------
Ran 5 tests in 0.019s

FAILED (failures=1)
```

4.2 容器化部署

本系统的后端使用 Docker 在生产环境部署，下方是构建使用的 Dockerfile 文件内容:

```
FROM python:3.10.0-buster

COPY requirements.txt /bank_service/
WORKDIR /bank_service/
RUN pip install -r requirements.txt -i https://mirrors.cloud.tencent.com/pypi/simple

FROM python:3.10.0-alpine3.15

COPY . /bank_service/
WORKDIR /bank_service/

COPY --from=0 /usr/local/lib/python3.10/site-packages/ \
```

```
    /usr/local/lib/python3.10/site-packages/

RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.aliyun.com/g' /etc/apk/repositories && \
    apk add gcc musl-dev python3-dev libffi-dev openssl-dev cargo && \
    pip install cryptography -i https://mirrors.cloud.tencent.com/pypi/simple && \
    apk del gcc musl-dev python3-dev libffi-dev openssl-dev cargo

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000", "--noreload"]
```

为了降低镜像体积，基础镜像使用预安装 `Python3.10.0` 的 `alpine` 轻量级 `Linux` 系统，因为后端使用了一些二进制包，需要在构建时编译，镜像系统中需要存在 `gcc` 等构建工具。为此，使用了 `Docker` 提供的多阶段构建策略。首先使用 `python:3.10.0-buster` 全量镜像执行 `pip install` 安装依赖。构建完成后，将镜像更换为 `python:3.10.0-alpine3.15` 轻量镜像，然后将前一阶段构建得到的 `site-packages` 转移到新镜像的对应位置。

通过多阶段构建，最终镜像的大小控制在 `110MB` 左右。

5 自动化

5.1 CI/CD

工程使用 `Github Actions` 提供的 `Workflow` 实现了一个基础 CI，服务器上使用 bash 脚本编写了一个拉取更新并构建镜像的简易 CD，提高了开发效率。

下方是 CD 脚本：

```
spawn git pull
except "Username" { send "PopChicken\n" }
expect "Password" { send "ghp_secret_key\n" }

docker rm bank_service --force
docker image rm bankservice
docker build -t bankservice:latest .
```