DSL

刘昱辉

2024 年 11 月 17 日

目录

第一章　概览

该项目实现了基于 DSL（Domain Specific Language）的客服机器人，提供了一种简单的语言，用于定义客服机器人的行为。

该项目采用 B/S 架构，包含了前端和后端两个模块，前端使用 React 框架，后端使用 C++ 实现。

提供了美观的 UI 界面，以及支持并行的客服机器人解释执行。

支持查询/修改 sqlite 数据库，支持限时输入，支持 CLI/浏览器多种交互方式。

第二章　模块

包含前后端两个模块。前后端通过WebSocket协议进行通信。

2.1　前端

前端采用React框架，使用NextJS框架进行渲染。使用TailwindCSS进行样式设计。使用Jest和Testing Library进行单元测试、测试桩。使用TypeDoc生成文档。

支持日间/夜间模式切换。

2.1.1　运行

```
# run frontend
cd frontend
# install dependencies
pnpm install # or: npm install
# run in development mode
pnpm dev # or: npm run dev
# build and run in production mode
pnpm build # or: npm run build
pnpm start # or: npm start
```

2.1.2　组件

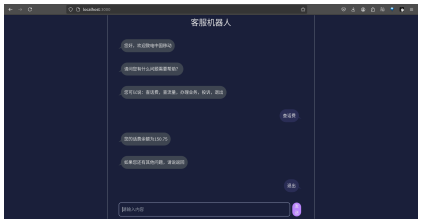
- `Home`：主界面，包含了 `MessageComposer`、`MessageDisplay`、`Title` 等组件。

- `MessageComposer`:用于编辑并发送消息。包含 `ClientBubble`、`ServerBubble` 等组件。
- `MessageDisplay`：用于显示消息。
- `Title`：用于显示标题。
- `ClientBubble`：用于显示客户端消息的气泡。
- `ServerBubble`：用于显示服务器消息的气泡。
- `Container`：用于包裹组件。

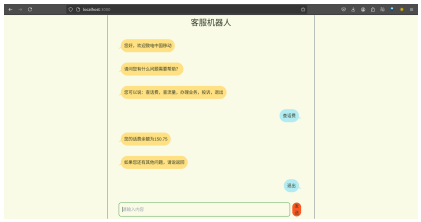
2.1.3　通信

前端通过 WebSocket 协议与后端通信，使用 `ws` 库。

2.1.4　截图



(a) 前端，夜间



(b) 前端，日间

图 2.1: 前端外观

2.2　后端

后端使用 C++ 实现，自主实现了词法分析、语法分析、语义分析、解释执行等功能。使用GoogleTest进行单元测试。使用Doxygen生成文档。

经过词法分析得到 Token 序列，然后通过递归下降分析器进行语法分析，得到抽象语法树，然后进行语义分析，检测错误并得到常量表和过程表，最后通过解释执行器进行解释执行。采用 visitor 设计模式来访问抽象语法树的结点。

使用异步 I/O 模型，支持并行处理多个客户端请求。使用条件变量和互斥量来实现限时输入。

2.2.1　运行

```
# first ensure boost is installed. See https://boost.org
# run backend
cd backend
cmake -S . -B build # if already exists, skip this step
cmake --build build # if already exists, skip this step
cd build
./DSL -m browser
```

更多命令行参数：使用`-h` 选项查看。

2.2.2　通信

后端通过 WebSocket 协议与前端通信，使用 `boost::asio` 和 `boost::beast` 库。

2.2.3　第三方库

- `boost`：用于实现 WebSocket 通信、解析命令行参数。
- `spdlog`：用于日志记录。
- `sqlite`：用于存储常量。

2.2.4　截图

```
~/projects/DSL/backend (main*) » ./build/DSL -m browser
[2024-11-16 19:20:16.502] [info] Successfully build AST.
[2024-11-16 19:20:16.502] [info] Running in browser mode.
[2024-11-16 19:20:16.502] [info] WebSocket Server listening on ws://localhost:8080
[2024-11-16 19:20:18.141] [info] New connection.
[2024-11-16 19:20:18.141] [info] Send: 您好，欢迎致电中国移动
[2024-11-16 19:20:18.141] [info] Send: 请问您有什么问题需要帮助？
[2024-11-16 19:20:18.141] [info] Send: 您可以说：查话费，查流量，办理业务，投诉，退出
[2024-11-16 19:20:20.566] [info] Received message: 话费
[2024-11-16 19:20:20.568] [info] Send: 您的话费余额为150.75
[2024-11-16 19:20:20.568] [info] Send: 如果您还有其他问题，请说返回
[2024-11-16 19:20:22.078] [info] Received message: 退出
[2024-11-16 19:20:22.078] [info] Send: 感谢您的来电，再会
[2024-11-16 19:20:22.078] [info] Program exited.
```

图 2.2: 后端

第三章　语法

以下是该语言的EBNF文法描述：

```
        program  ::= {constant-define} {procedure}
constant-define  ::= id "=" string
      procedure  ::= "procedure" id "{" {statement} "}"
      statement  ::= say | listen | lookup | exit | id
            say  ::= "say" id | string {"," id | string}
         listen  ::= "listen" "timelimit" number "{"
                     {has | timeout | default | anything} "}"
         lookup  ::= "lookup" id "in" id ":" id
       anything  ::= "anything" "?" id | writeto
            has  ::= "has" id | string "?" id | writeto
        timeout  ::= "timeout" "?" id | writeto
        default  ::= "default" "?" id | writeto
        writeto  ::= "writeto" id ":" id ":" id
         string  ::= '"' {allcharacters - '"'} '"'
             id  ::= [a - zA - Z] id | [a - zA - Z]
  allcharacters  ::=? all visible ASCII characters ?
```

除此之外，还有以下 2 点约束：

1. 必须有 main 过程；

2. 过程名不能重复；常量名不能重复。

DSL 的设计原则是最简化，只包含领域特定的语言特性，不包含通用的编程语言特性。

使用 Parser 类进行语法分析,得到抽象语法树 shared_ptr<Program>。可以用 Printer 类打印抽象语法树：

Printer printer; program->accept(printer);。

以下是说明：

- 包含常量定义和过程定义两个部分，常量定义部分在前；
- 常量定义部分包含了常量名和常量值，例如：greeting = "Hello, world!";；
- 过程定义部分包含了过程名和过程体，例如：procedure main { say "Hello, world!" }；
- 过程体包含了多个语句，语句种类有：say、listen、lookup、exit、jump；
- say 语句用于输出消息，可以输出常量或字符串，以逗号分隔，例如：say greeting, "Tom"；
- listen 语句用于接收消息,可以设置超时时间,以及设置 has、timeout、default、anything 等子句；例如：listen timelimit 10 { default ? defaultProc }；
- lookup 语句用于查询数据库,例如:lookup balance in db:UserInfo;；其中 db 是数据库名，用一个常量指定，UserInfo 是表名，balance 是字段名；
- exit 语句用于退出程序；
- jump 语句用于跳转到指定过程,直接用过程名即可,例如:defaultProc；
- anything 子句用于接收任意输入，例如：anything ? defaultProc；
- has 子句用于接收指定输入，例如：has "pattern" ? defaultProc；

- `timeout` 子句用于超时时执行，例如：`timeout ? defaultProc`；
- `default` 子句用于默认执行，例如：`default ? defaultProc`；
- 子句执行的动作可以是跳转到指定过程，也可以是写入数据库，例如：`writeto db:UserInfo:balance`，其中 `db` 是数据库名，`UserInfo` 是表名，`balance` 是字段名。

第四章　测试

4.1　前端

前端测试使用 Jest 框架和 Testing Library。为每个组件编写了单元测试，以及为前后端通信编写了测试桩。测试文件位于 `frontend/test` 目录下。如下：

单元测试：

`bubble.test.tsx`

`container.test.tsx`

`message_composer.test.tsx`

`message_display.test.tsx`

`title.test.tsx`

测试桩：

`page.test.tsx`

自动测试脚本：

```
# test frontend
cd frontend
pnpm test # or: npm test
```

测试结果如下：

```
Test Suites: 6 passed, 6 total
Tests:       17 passed, 17 total
Snapshots:   0 total
Time:        1.306 s

Ran all test suites.
```

测试覆盖率见图 4.1。

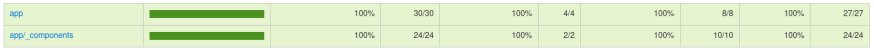


图 4.1: 测试覆盖率

4.2　后端

后端测试使用 GoogleTest 框架。为每个模块编写了单元测试。为前后端通信编写了测试桩。

测试文件位于 `backend/test` 目录下。测试桩位于 `backend/test/stub` 目录下。

测试脚本：

```
# test frontend
cd backend
cmake -S . -B build # if already exists, skip this step
cmake --build build # if already exists, skip this step
cd build
ctest # or: to run single test or no ctest: ./test/lexer_test
```

测试结果如下：

```
100% tests passed, 0 tests failed out of 32
```

第五章　得分标准完成情况

5.1　风格

5.1.1　注释

- 前端：TypeDoc 风格，对每个组件进行了注释。
- 后端：Doxygen 风格，对文件、类、函数进行了注释。

5.1.2　命名

- 前端：CamelCase，TypeScript 的惯用命名法。
- 后端：
  - Class: CamelCase
  - Enum: CamelCase
  - Function: camelBack
  - GlobalConstant: UPPER_CASE
  - Member: lower_case
  - Variable: camelBack

5.1.3　其他

- 前端：通过了 eslint 的检查，使用了 prettier 格式化。
- 后端：通过了 clang-tidy、CLion 内置 linter 的检查，使用 clang-format 格式化。

5.2　设计和实现

5.2.1　数据结构

- 后端：Lexer，Parser，AST，Interpreter，SemanticAnalysis 等类。

5.2.2　模块划分

- 前端：Component、MessageComposer、MessageDisplay、Bubble 等。
- 后端：lexer，parser，interpreter，network，sqlite interface，argument parser 等。

5.2.3　功能

- 前端：light/dark mode，自动滑动到最新消息，发送/接收。
- 后端：network，读/写 sqlite 数据库，并行处理等。

5.2.4　文档

- 前端：TypeDoc 生成，见附录 A。
- 后端：Doxygen 生成，见附录 A。

5.3　接口

5.3.1　程序间接口

- 前后端：WebSocket 协议，此协议为双向通信协议，支持并行处理多个客户端请求，是即时通信的常用协议。

5.3.2　人机接口

- 两套：CLI 和浏览器，CLI 支持命令行参数，浏览器支持图形界面。

5.4　测试

5.4.1　前端

- 测试桩：`test/page.text.tsx`。
- 自动测试脚本：`pnpm test`。

5.4.2　后端

- 测试桩：`test/stub`。
- 自动测试脚本：`ctest`。

5.5　记法

使用 EBNF 完整、准确地描述了 DSL 的语法。见 三。

第六章　总结

该项目实现了基于 DSL 的客服机器人。通过该项目，我学会了 B/S 架构的开发，掌握了风格、设计、实现、接口、测试、文档等方面的知识。

附录 A　文档

A.1　前端

首页如图 A.1。参见前端文档。如果无法打开，请查看 `frontend/docs/index.html`。

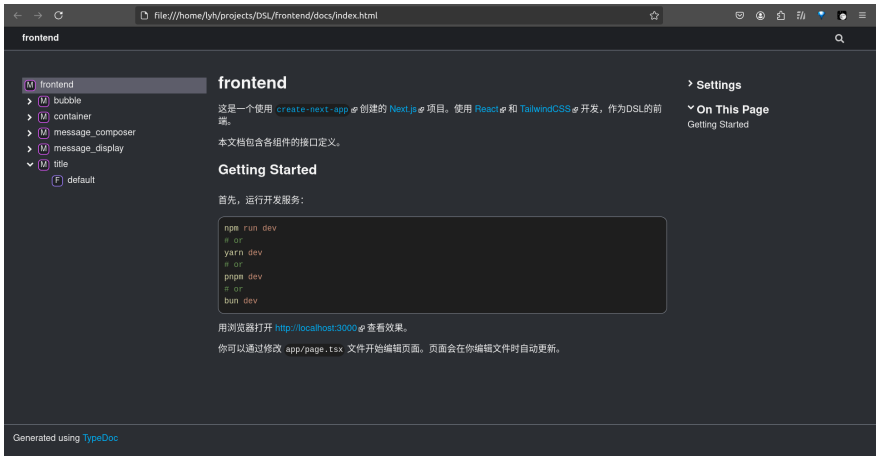


图 A.1: 前端文档

A.2　后端

首页如图 A.2。参见后端文档。如果无法打开，请查看 `backend/doc/html/index.html`。

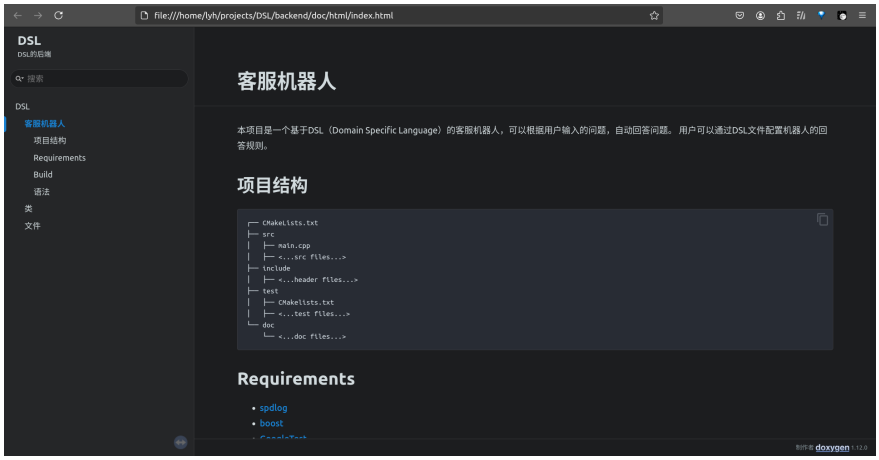

file:///home/lyh/projects/DSL/backend/doc/html/index.html
DSL
DSL的后端
搜索
DSL
客服机器人
项目结构
Requirements
Build
语法
类
文件
客服机器人
本项目是一个基于DSL（Domain Specific Language）的客服机器人，可以根据用户输入的问题，自动回答问题。 用户可以通过DSL文件配置机器人的回答规则。
项目结构
├── CMakeLists.txt
├── src
│ ├── main.cpp
│ ├── <...src files...>
├── include
│ ├── <...header files...>
├── test
│ ├── CMakeLists.txt
│ ├── <...test files...>
└── doc
└── <...doc files...>
Requirements
spdlog
boost
doxygen

图 A.2: 后端文档