Architecture Guide

Tổng quan về quy trình nhận diện action

Trong dự án, **action** là ý định (intent) của lệnh người dùng nhập, ví dụ:

- Lệnh "đặt lịch hẹn vào thứ Hai lúc 10 giờ sáng" → Action: schedule (đặt lịch).
- Lệnh "hủy lịch hẹn vào thứ Ba" → Action: cancel (hủy lịch).

Quy trình nhận diện action bao gồm các bước chính:

1. **Phân tích cú pháp lệnh** (sử dụng parser.py và các file sinh ra từ ANTLR4): Chuyển lệnh dạng văn bản thành cây cú pháp (parse tree).
2. **Xử lý cây cú pháp** (sử dụng processor.py và CommandVisitor): Duyệt cây cú pháp để xác định action và trích xuất thông tin liên quan (như ngày, giờ).
3. **Trả về kết quả**: Kết quả là một cấu trúc dữ liệu (ví dụ: dictionary) chứa action và các tham số (như {'intent': 'schedule', 'day': 'thứ Hai', 'time': '10 giờ sáng'}).

Tôi sẽ giải thích từng bước và liên hệ với mã trong dự án.

Bước 1: Phân tích cú pháp lệnh (Parser)

1.1. Vai trò của Command.g4

File src/grammar/Command.g4 định nghĩa ngữ pháp cho các lệnh người dùng. Đây là nơi quy định cấu trúc của các lệnh và cách chúng được phân tích. Nội dung mẫu của Command.g4 (dựa trên hướng dẫn trước):

```
grammar Command;

command: schedule | cancel ;
schedule: 'đặt' 'lịch' 'hẹn' 'vào' day 'lúc' time ;
cancel: 'hủy' 'lịch' 'hẹn' 'vào' day ;
day: 'thứ Hai' | 'thứ Ba' | 'thứ Tư' | 'thứ Năm' | 'thứ Sáu' | 'thứ Bảy' | 'Chủ N
hật' ;
time: NUMBER 'giờ' ('sáng' | 'chiều')? ;
```

```
NUMBER: [0-9]+ ;
WS: [ \t\r\n]+ → skip ;
```

Giải thích:

- Quy tắc command định nghĩa hai loại lệnh: schedule (đặt lịch) và cancel (hủy lịch).
- Quy tắc schedule yêu cầu chuỗi từ khóa cố định (đặt lịch hẹn vào) theo sau là day (ngày) và time (thời gian).
- Quy tắc cancel chỉ yêu cầu day.
- Các token như NUMBER (số) và WS (khoảng trắng) hỗ trợ phân tích cú pháp.

1.2. Tạo mã từ Command.g4

Script scripts/generate_parser.sh sử dụng file JAR lib/antlr-4.9.2-complete.jar để tạo các file Python trong src/generated:

```
#!/bin/bash
java -jar lib/antlr-4.9.2-complete.jar -Dlanguage=Python3 -visitor -o src/ge
nerated src/grammar/Command.g4
```

Các file được tạo:

- **CommandLexer.py:** Phân tích lệnh thành danh sách token (ví dụ: "đặt", "lịch", "thứ Hai", v.v.).
- **CommandParser.py:** Tạo cây cú pháp từ danh sách token, dựa trên quy tắc trong Command.g4.
- **CommandVisitor.py:** Cung cấp lớp cơ sở để duyệt cây cú pháp và xử lý các quy tắc (như schedule, cancel).

Lưu ý: Tùy chọn -visitor đảm bảo CommandVisitor.py được tạo, cần thiết để nhận diện action.

1.3. Vai trò của parser.py

File src/core/parser.py sử dụng các file sinh ra để phân tích lệnh người dùng thành cây cú pháp. Nội dung mẫu:

```
from antlr4 import *
from src.generated.CommandLexer import CommandLexer
from src.generated.CommandParser import CommandParser

def parse_command(command_text):
    # Chuyển lệnh thành luồng ký tự
    input_stream = InputStream(command_text)
    # Tạo lexer để phân tích token
    lexer = CommandLexer(input_stream)
    # Tạo luồng token
    token_stream = CommonTokenStream(lexer)
    # Tạo parser để phân tích cú pháp
    parser = CommandParser(token_stream)
    # Phân tích từ quy tắc gốc 'command'
    tree = parser.command()
    return tree
```

Quy trình trong parser.py:

1. **Nhận đầu vào**: Lệnh người dùng (ví dụ: "đặt lịch hẹn vào thứ Hai lúc 10 giờ sáng").
2. **Lexer**: CommandLexer chia lệnh thành token:
    - "đặt", "lịch", "hẹn", "vào", "thứ Hai", "lúc", "10", "giờ", "sáng".
3. **Parser**: CommandParser sử dụng token để tạo cây cú pháp, dựa trên quy tắc command trong Command.g4. Kết quả là một cây cú pháp với nút gốc là command, có thể là nhánh schedule hoặc cancel.
4. **Trả về**: Cây cú pháp (tree) được truyền sang bước xử lý tiếp theo.

Ví dụ cây cú pháp (cho lệnh "đặt lịch hẹn vào thứ Hai lúc 10 giờ sáng"):

- Nút gốc: command
- Nhánh con: schedule
    - Con của schedule: day ("thứ Hai"), time ("10 giờ sáng")

Bước 2: Xử lý cây cú pháp (Processor)

2.1. Vai trò của processor.py

File src/core/processor.py duyệt cây cú pháp để xác định **action** và trích xuất thông tin (như ngày, giờ). Nó sử dụng lớp CommandVisitor từ CommandVisitor.py. Nội dung mẫu:

```
from antlr4 import *
from src.generated.CommandVisitor import CommandVisitor
from src.generated.CommandParser import CommandParser

class CommandProcessor(CommandVisitor):
    def visitSchedule(self, ctx: CommandParser.ScheduleContext):
        day = ctx.day().getText()
        time = ctx.time().getText()
        return {"intent": "schedule", "day": day, "time": time}

    def visitCancel(self, ctx: CommandParser.CancelContext):
        day = ctx.day().getText()
        return {"intent": "cancel", "day": day}
```

Giải thích:

- CommandProcessor kế thừa CommandVisitor, một lớp cơ sở được tạo bởi ANTLR4 trong CommandVisitor.py.
- Mỗi phương thức visitX tương ứng với một quy tắc trong Command.g4:
    - visitSchedule: Xử lý nhánh schedule, trích xuất day và time, trả về dictionary với intent: "schedule".
    - visitCancel: Xử lý nhánh cancel, trích xuất day, trả về dictionary với intent: "cancel".
- ctx (context) là đối tượng chứa thông tin về nhánh cây cú pháp, cho phép truy cập các thành phần như day() hoặc time().

2.2. Cách nhận diện action

- **Bước duyệt cây cú pháp**:
    - Khi CommandProcessor được gọi với cây cú pháp từ parser.py, nó tự động duyệt cây và gọi phương thức tương ứng dựa trên nhánh:

    - Nếu cây có nhánh schedule, visitSchedule được gọi → Action là schedule.
    - Nếu cây có nhánh cancel, visitCancel được gọi → Action là cancel.
- **Trích xuất thông tin**:
    - Trong visitSchedule, ctx.day().getText() lấy văn bản của quy tắc day (ví dụ: "thứ Hai").
    - Tương tự, ctx.time().getText() lấy văn bản của quy tắc time (ví dụ: "10 giờ sáng").
    - Kết quả là một dictionary chứa action (intent) và các tham số.

Ví dụ:

- Lệnh: "đặt lịch hẹn vào thứ Hai lúc 10 giờ sáng"
- Cây cú pháp: Nhánh schedule với day = "thứ Hai", time = "10 giờ sáng"
- visitSchedule trả về:python

```
{"intent": "schedule", "day": "thứ Hai", "time": "10 giờ sáng"}
```

Bước 3: Tích hợp trong CLI

File src/interfaces/cli.py gọi parse_command (từ parser.py) và CommandProcessor (từ processor.py) để xử lý lệnh người dùng. Nội dung mẫu:

```
from src.core.parser import parse_command
from src.core.processor import CommandProcessor

def main():
    print("Nhập lệnh (nhập 'thoát' để dừng):")
    while True:
        cmd = input("Lệnh: ")
        if cmd.lower() == "thoát":
            break
        try:
            tree = parse_command(cmd)
            processor = CommandProcessor()
            result = processor.visit(tree)
```

```
            print("Kết quả:", result)
        except Exception as e:
            print("Lỗi:", str(e))

if __name__ == "__main__":
    main()
```

Quy trình trong cli.py:

1. Nhận lệnh từ người dùng (ví dụ: "đặt lịch hẹn vào thứ Hai lúc 10 giờ sáng").
2. Gọi parse_command để tạo cây cú pháp.
3. Tạo đối tượng CommandProcessor và gọi visit(tree) để duyệt cây, nhận diện action, và trích xuất thông tin.
4. In kết quả (ví dụ: {'intent': 'schedule', 'day': 'thứ Hai', 'time': '10 giờ sáng'}).

Minh họa toàn bộ quy trình

Hãy xem cách lệnh "đặt lịch hẹn vào thứ Hai lúc 10 giờ sáng" được xử lý:

1. **Đầu vào**: Người dùng nhập lệnh trong CLI.
2. **Lexer (CommandLexer)**:
   - Chia lệnh thành token: "đặt", "lịch", "hẹn", "vào", "thứ Hai", "lúc", "10", "giờ", "sáng".
3. **Parser (CommandParser)**:
   - So khớp token với quy tắc schedule trong Command.g4.
   - Tạo cây cú pháp với nhánh schedule, chứa day = "thứ Hai", time = "10 giờ sáng".
4. **Processor (CommandProcessor)**:
   - Duyệt cây, gọi visitSchedule.
   - Trích xuất day và time, trả về:python

     ```
     {"intent": "schedule", "day": "thứ Hai", "time": "10 giờ sáng"}
     ```
5. **CLI**:
   - In kết quả ra màn hình.

Tương tự, lệnh "hủy lịch hẹn vào thứ Ba" sẽ kích hoạt visitCancel, trả về:

python

```
{"intent": "cancel", "day": "thứ Ba"}
```

Tại sao cách này hiệu quả?

1. **Cấu trúc ngữ pháp rõ ràng**:
   - Command.g4 định nghĩa chính xác cấu trúc lệnh, giúp ANTLR4 phân tích đúng cú pháp.
   - Các quy tắc như schedule và cancel ánh xạ trực tiếp đến các action.
2. **Mô hình Visitor mạnh mẽ**:
   - CommandVisitor cho phép xử lý từng nhánh cây cú pháp một cách riêng biệt, dễ dàng xác định action và trích xuất dữ liệu.
   - Mỗi phương thức visitX tương ứng với một action, giúp mã dễ bảo trì.
3. **Tính mô-đun**:
   - parser.py tách biệt việc phân tích cú pháp.
   - processor.py tách biệt việc nhận diện action và xử lý dữ liệu.
   - cli.py chỉ tập trung vào giao tiếp với người dùng.

Các lỗi tiềm ẩn và cách xử lý

1. **Lệnh không khớp ngữ pháp**:
   - Nếu người dùng nhập "đặt lịch vào thứ Hai" (thiếu "hẹn" hoặc "lúc"), parser sẽ báo lỗi.
   - **Giải pháp**: Thêm xử lý lỗi trong cli.py:python

     ```
     except Exception as e:
         print("Lệnh không hợp lệ. Ví dụ: đặt lịch hẹn vào thứ Hai lúc 10 giờ sáng")
     ```
2. **Không tạo CommandVisitor.py**:
   - Nếu thiếu tùy chọn -visitor trong generate_parser.sh, processor.py sẽ báo lỗi ModuleNotFoundError: No module named 'CommandVisitor'.

- **Giải pháp**: Đảm bảo script có -visitor (đã hướng dẫn trước).

3. **Không tương thích phiên bản ANTLR4**:
   - Mã trong processor.py có thể được viết cho ANTLR4 4.13.1, gây lỗi với 4.9.2.
   - **Giải pháp**: Kiểm tra tài liệu ANTLR4 4.9.2 hoặc nâng cấp lên 4.13.1 (xem hướng dẫn trước).
4. **Lỗi import**:
   - Nếu thiếu __init__.py trong src/generated, import CommandVisitor sẽ thất bại.
   - **Giải pháp**: Đảm bảo __init__.py tồn tại (đã hướng dẫn trong README)

Tổng kết

Cách dự án nhận diện action thông qua command dựa trên:

1. **Phân tích cú pháp**:
   - Command.g4 định nghĩa cấu trúc lệnh.
   - parser.py sử dụng CommandLexer và CommandParser để tạo cây cú pháp.
2. **Xử lý action**:
   - processor.py sử dụng CommandVisitor để duyệt cây, gọi phương thức tương ứng (visitSchedule, visitCancel) để xác định action và trích xuất dữ liệu.
3. **Tích hợp**:
   - cli.py kết nối parser và processor, hiển thị kết quả cho người dùng.

Ví dụ:

- Lệnh: "đặt lịch hẹn vào thứ Hai lúc 10 giờ sáng"
- Quy trình: Lexer → Parser (tạo cây với nhánh schedule) → Processor (visitSchedule) → Kết quả: {'intent': 'schedule', 'day': 'thứ Hai', 'time': '10 giờ sáng'}.