GoF · 13/23

ПОВЕДЕНЧЕСКИЙ

Python C++23 Rust Java



Цепочка обязанностей

Chain of Responsibility

Алексей Примак

Алексей Примак

Цепочка обязанностей. Паттерн GoF на Python, C++23, Rust и Java

https://litres.ru/74810822
SelfPub; 2026

Аннотация

Не каждый запрос должен обрабатываться одним и тем же компонентом. «Цепочка обязанностей» показывает, как передавать запрос по цепочке обработчиков, пока подходящий участник не сможет его обработать. На примере службы поддержки разобраны структура паттерна GoF, порядок передачи и варианты настройки цепочки. Затем идут реализации на Python, C++23, Rust и Java, модификации и самостоятельные задания.

Содержание

Введение 4
Как читать 5
Глава 13. Реализация на Python 6
13.1. Зачем нужна цепочка обязанностей 6
13.2. Участники и схема 7
13.3. Классический пример: служба поддержки 8
Конец ознакомительного фрагмента. 13

Цепочка обязанностей. Паттерн GoF на Python, C++23, Rust и Java

Введение

Эта книга посвящена одному паттерну проектирования — **Цепочка обязанностей**(Chain of Responsibility). Он относится к каталогу «Банды четырёх» (Gang of Four, GoF). Здесь паттерн разобран сразу в четырёх реализациях: на Python, C++23, Rust и Java. Каждая реализация — отдельная часть книги: сначала задача и общая схема, затем код, модификации, гибридные варианты и самостоятельные работы.

Как читать

Основы каждого языка здесь не повторяются: они собраны в книгах серий по соответствующему языку. Разделы каждой части нумеруются заново, поэтому ссылка «раздел 1.2» указывает на раздел той же части, где она сделана.

Глава 13. Реализация на Python

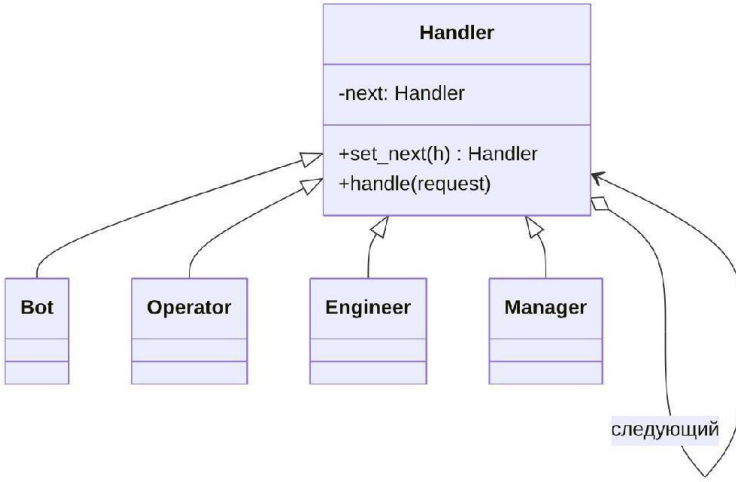
13.1. Зачем нужна цепочка обязанностей

В службу поддержки приходит обращение. Простой вопрос решает бот, вопрос посложнее — оператор первой линии, технический — инженер, а вопрос о возврате крупной суммы — только руководитель. Отправитель не должен знать, кто именно ответит: он просто отправляет запрос.

Цепочка обязанностей передаёт запрос по цепочке обработчиков. Каждый решает: обработать запрос самому, передать дальше или сделать и то и другое. Отправитель знает только первое звено. Обработчики можно добавлять, убирать и переставлять, не трогая ни отправителя, ни друг друга.

Вы встречали этот паттерн много раз: промежуточные слои (middleware) веб-фреймворков, обработчики модуля logging, всплытие событий в браузере, блоки except при поиске подходящего обработчика исключения.

13.2. Участники и схема



Handler— интерфейс обработки и ссылка на следующего.

Concrete Handler— обрабатывает свои запросы, остальные передаёт дальше.

Client— отправляет запрос первому звену.

13.3. Классический пример: служба поддержки

Файл 01_support.py:

01_support.py Python

```python
from __future__ import annotations

from dataclasses import dataclass


@dataclass
class Ticket:
    topic: str
    text: str
    amount: int = 0


class Handler:
    def __init__(self) -> None:
        self._next: Handler | None = None

    def set_next(self, handler: Handler) -> Handler:
        self._next = handler
        return handler

    def handle(self, t: Ticket) -> str:
        if self._next is None:
            return "не обработано: передано в общий чат"
        return self._next.handle(t)


class Bot(Handler):
    FAQ = {"пароль": "сбросьте пароль по ссылке «Забыли пароль?»
        ", "доставка": "доставка 2–5 дней"}

    def handle(self, t: Ticket) -> str:
        for key, answer in self.FAQ.items():
            if key in t.text.lower():
                return f"бот: {answer}"
```

Листинг 13.1 *— Классический пример: служба поддерж-*

ки

01_support.py (продолжение) Python

```
        return super().handle(t)


class Operator(Handler):
    def handle(self, t: Ticket) -> str:
        if t.topic in {"заказ", "аккаунт"}:
            return f"оператор: разобрался с «{t.text}»"
        return super().handle(t)


class Engineer(Handler):
    def handle(self, t: Ticket) -> str:
        if t.topic == "ошибка":
            return f"инженер: завёл задачу по «{t.text}»"
        return super().handle(t)


class Manager(Handler):
    def __init__(self, limit: int) -> None:
        super().__init__()
        self.limit = limit

    def handle(self, t: Ticket) -> str:
        if t.topic == "возврат":
            verdict = "одобрил" if t.amount <= self.limit else "
                отправил в финотдел"
            return f"руководитель: {verdict} возврат {t.amount}
                ₽"
        return super().handle(t)


if __name__ == "__main__":
    chain = Bot()
    chain.set_next(Operator()).set_next(Engineer()).set_next(
```

Листинг 13.1 (продолжение) *— Классический пример: служба поддержки*

01_support.py (продолжение) Python

```
            Manager(limit=10_000))
    tickets = [
        Ticket("аккаунт", "Не помню пароль"),
        Ticket("заказ", "Где мой заказ 1532?"),
        Ticket("ошибка", "Кнопка оплаты не нажимается"),
        Ticket("возврат", "Верните деньги", 4_500),
        Ticket("возврат", "Верните деньги за ноутбук", 85_000),
        Ticket("сотрудничество", "Хотим рекламу"),
    ]
    for t in tickets:
        print(chain.handle(t))
```

```
$ python 01_support.py
бот: сбросьте пароль по ссылке «Забыли пароль?»
оператор: разобрался с «Где мой заказ 1532?»
инженер: завёл задачу по «Кнопка оплаты не нажимается»
руководитель: одобрил возврат 4500 ₽
руководитель: отправил в финотдел возврат 85000 ₽
не обработано: передано в общий чат
```

Листинг 13.1 (продолжение) *— Классический пример: служба поддержки*

Разбор.Каждый обработчик проверяет «своё» и иначе вызывает super().handle(t), который передаёт запрос следующему. Последнее звено базового класса возвращает ответ по умолчанию, поэтому ни одно обращение не теряется.

set_next возвращает переданный обработчик, что позволяет строить цепочку одной строкой. Первое обращение попало в тему «аккаунт», которую обслуживает оператор, но бот стоит раньше и ответил сам, найдя ключевое слово: порядок звеньев определяет приоритет.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, купив полную легальную версию на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.