GoF · 16/23

ПОВЕДЕНЧЕСКИЙ

Python C++23 Rust Java



Итератор

Iterator

Алексей Примак

Алексей Примак

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

«Автор»
2026

Примак А.

Итератор. Паттерн GoF на Python, C++23, Rust и Java /
А. Примак — «Автор», 2026

Клиенту часто нужно просто последовательно получить элементы коллекции, не зная, как они хранятся внутри. «Итератор» показывает, как отделить способ обхода от структуры данных и поддерживать разные независимые обходы. Паттерн GoF разобран на примерах списков, деревьев, графов и других коллекций, а затем реализован на Python, C++23, Rust и Java. В книге также есть модификации и задания для практики.

© Примак А., 2026
© Автор, 2026

Содержание

Введение 5
Как читать 6
Глава 16. Реализация на Python 7
16.1. Зачем нужен итератор 7
16.2. Участники и схема 8
16.3. Классический пример: плейлист 11
Конец ознакомительного фрагмента. 13

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

Введение

Эта книга посвящена одному паттерну проектирования — **Итератор**(Iterator). 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 16. Реализация на Python

16.1. Зачем нужен итератор

Коллекция может быть списком, деревом, графом, файлом на диске или страницами ответа API. Клиенту же обычно нужно одно: перебрать элементы по одному, не зная, как они хранятся. Кроме того, иногда нужно несколько независимых обходов одновременно или разные порядки обхода одной структуры.

Итератор выносит обход из коллекции в отдельный объект. Он помнит текущую позицию и умеет выдавать следующий элемент. Коллекция только создаёт итераторы.

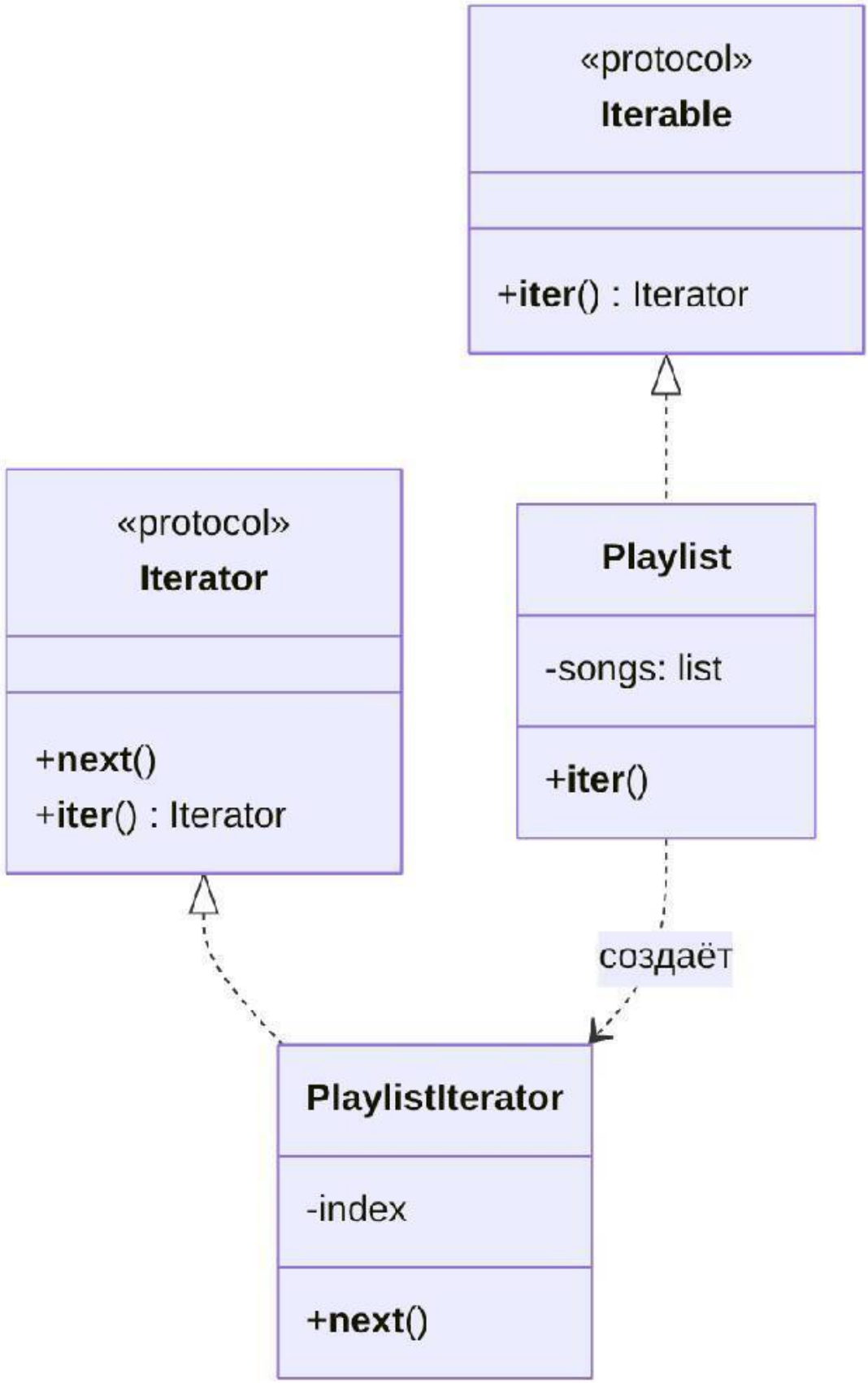
В Python итератор встроен в язык глубже, чем любой другой паттерн. Цикл for, распаковка, list(), sum(), генераторные выражения — всё работает через протокол итерации:

итерируемый объект имеет метод __iter__, возвращающий итератор;

итератор имеет метод __next__, возвращающий следующий элемент или выбрасывающий StopIteration, и __iter__, возвращающий себя.

Генераторы (yield) — самый удобный способ написать итератор, а модуль itertools — набор готовых итераторов-кирпичиков.

16.2. Участники и схема


«protocol»
Iterable
+iter() : Iterator
«protocol»
Iterator
+next()
+iter() : Iterator
Playlist
-songs: list
+iter()
создаёт
PlaylistIterator
-index
+next()

Iterator— интерфейс перебора, хранит позицию.
Concrete Iterator— реализует обход конкретной структуры.
Iterable (**Aggregate**)— создаёт итераторы.

16.3. Классический пример: плейлист

Файл 01_playlist.py:

01_playlist.py — Python

```python
from __future__ import annotations

from typing import Iterator


class Playlist:
    def __init__(self, *songs: str) -> None:
        self._songs = list(songs)

    def __iter__(self) -> PlaylistIterator:
        return PlaylistIterator(self._songs)

    def reversed(self) -> PlaylistIterator:
        return PlaylistIterator(self._songs, step=-1)


class PlaylistIterator:
    def __init__(self, songs: list[str], step: int = 1) -> None:
        self._songs, self._step = songs, step
        self._i = 0 if step > 0 else len(songs) - 1

    def __iter__(self) -> Iterator[str]:
        return self

    def __next__(self) -> str:
        if not 0 <= self._i < len(self._songs):
            raise StopIteration
        song = self._songs[self._i]
        self._i += self._step
        return song


if __name__ == "__main__":
    pl = Playlist("Кино — Группа крови", "Аквариум — Город", "
```

Листинг 16.1 *— Классический пример: плейлист*

01_playlist.py (продолжение) Python

```
        ДДТ — Осень")
    for n, song in enumerate(pl, 1):
        print(n, song)
    print("Наоборот:", list(pl.reversed()))
    a, b = iter(pl), iter(pl)  # два независимых обхода
    print(next(a), "|", next(a), "|", next(b))
    it = iter(pl)
    print("Первый проход:", len(list(it)), "| второй по тому же
        итератору:", len(list(it)))
```

```
$ python 01_playlist.py
1 Кино — Группа крови
2 Аквариум — Город
3 ДДТ — Осень
Наоборот: ['ДДТ — Осень', 'Аквариум — Город', 'Кино — Группа
    крови']
Кино — Группа крови | Аквариум — Город | Кино — Группа крови
Первый проход: 3 | второй по тому же итератору: 0
```

Листинг 16.1 (продолжение) *— Классический пример: плейлист*

Разбор. Плейлист итерируемый: каждый вызов iter(pl) создаёт новый итератор со своей позицией, поэтому обходы a и b независимы. Итератор одноразовый: после исчерпания он больше ничего не выдаёт. Это ключевое различие между итерируемым объектом и итератором. Обратный порядок — просто другой итератор над теми же данными.

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.