Парадигмы программирования

Александра Волосова,
к.т.н., доцент кафедры
ИУ5

Презентация разработана в рамках гранта «Грант на обучение студентов по образовательным программам высшего образования для топ-специалистов в сфере информационных технологий. Договор № 70-2025-000850 с АНО «Аналитический центр при Правительстве Российской Федерации»

План лекции

1. Что такое парадигма программирования?
2. Императивное программирование
3. Структурное программирование
4. Процедурное программирование
5. Объектно-ориентированное программирование (ООП)
6. Декларативное программирование
7. Функциональное программирование
8. Логическое программирование
9. Событийно-ориентированное программирование
10. Метапрограммирование
11. Сравнительный анализ парадигм
12. Гибридные подходы
13. Заключение и выводы

Что такое парадигма программирования?

Определение 1: Способ концептуализации и структурирования процесса вычислений

Определение 2. Парадигма программирования — это совокупность идей и понятий, определяющих стиль написания программ и подход к решению задач

Ключевые аспекты:

- Это не язык, а стиль мышления и организации кода
- Языки могут поддерживать multiple paradigms (Python, JavaScript, Scala)
- Определяет как разработчик формулирует решение проблем

Императивный стиль (процедурный) — Уровень физических операций

Инженер-конструктор управляет каждым компонентом:

- Подать напряжение 12V на контакт A7 модуля питания
- Установить регистр 0x3F в значение 0b11001010
- Активировать реле номер 3 на 200ms
- Считать данные с датчика температуры по шине I2C 5. Если температура > 45°C, уменьшить мощность двигателя на 15%
- Отправить пакет данных через UART-порт

ООП — уровень абстракций и инкапсуляции

```
// Абстракция
class КосмическийАппарат {
private:
        НавигационнаяСистема навигация;
public:
        void выполнитьКоррекциюОрбиты() {
// Внутри вызываются нужные методы } };
// Использование
        КосмическийАппарат аппарат;
        аппарат.выполнитьКоррекциюОрбиты();
```

Функциональный стиль - математическая модель

```
# Чистые функции для преобразования данных
рассчитать_траекторию = lambda данные:    transform(данные, физические_законы)
обработать_телеметрию = compose(фильтровать, нормализовать, анализировать)
# Поток данных через преобразования
raw_data → обработать_телеметрию → рассчитать_траекторию → принять_решение
```

Создание **математической модели** полета, где система описывается как **композиция функций** преобразования данных.

Декларативный стиль – специфика требований

```
-- SQL-подобный декларативный подход
SELECT траектория, уровень_топлива
FROM состояние_аппарата
WHERE температура < 45.0
                AND связь_с_Землей = true
ORDER BY приоритет LIMIT 1;
```

Формулировка **научных требований** к миссии: *"Обеспечить стабильную орбиту с минимальным расходом топлива при безопасной температуре"* без указания **как** именно это достичь.

Событийно-ориентированный стиль

```
// Реакция на события
аппарат.на('перегрев', (температура) => {    двигатель.уменьшитьМощность();
                охлаждение.активировать(); });
аппарат.на('потеря_связи', () => {
                перейтиВАвтономныйРежим();
                сохранитьДанныеВБуфер(); });
```

Система **обратной связи**, где аппарат **реагирует** на изменения условий, как живой организм.

Стиль	Научная аналогия	Уровень абстракции
Императивный	Дифференциальные уравнения (шаг за шагом)	Низкий (физика)
ООП	Теория систем (взаимодействие компонентов)	Средний (инженерия)
Функциональный	Математические преобразования (функции)	Высокий (математика)
Декларативный	Формальная спецификация (требования)	Очень высокий (теория)
Событийный	Теория управления (обратная связь)	Динамический

Императивное программирование

Программа как последовательность команд, изменяющих состояние программы

Ключевые концепции:
- Переменные (состояние)
- Операторы присваивания (изменение состояния)
- Конструкции управления потоком (if, for, while)

Пример языков: C, Fortran, BASIC, Python (частично)

Фокус на том КАК достичь результата через последовательность шагов

Пример императивного кода (C)

```
#include <stdio.h>

int main() {
    int n = 5;
    int factorial = 1;

    for (int i = 1; i <= n; ++i) {
        factorial *= i;
    }

    printf("Factorial of %d is %d\n", n, factorial);
    return 0;
}
```

Структурное программирование

Развитие императивного подхода с акцентом на структуру

Принципы:
- Последовательность
- Ветвление (if/else)
- Циклирование (while, for)
- Отказ от goto

Теорема Бома-Якопини: Любая программа может быть написана с использованием только этих трех структур

Структурное программирование улучшает читаемость и поддерживаемость кода

Процедурное программирование

Организация кода вокруг процедур/функций

Ключевые концепции:
- Разделение программы на функции
- Локализация functionality
- Повторное использование кода
- Модульность

Пример языков: Pascal, C, Fortran

Теория: Процедурное программирование уменьшает дублирование кода и улучшает организацию

Пример процедурного кода (Pascal)

```
program Example;

function CalculateFactorial(n: Integer): Integer;
var
  i, fact: Integer;
begin
  fact := 1;
  for i := 1 to n do
    fact := fact * i;
  CalculateFactorial := fact;
end;

begin
  Writeln('Factorial: ', CalculateFactorial(5));
end.
```

Объектно-ориентированное программирование (ООП)

Организация программы вокруг объектов и их взаимодействия

Четыре столпа ООП:
- Инкапсуляция: Сокрытие реализации
- Наследование: Повторное использование кода
- Полиморфизм: Разные реализации интерфейсов
- Абстракция: Упрощение сложности

Пример языков: Java, C++, Python, C#

Теория: ООП моделирует реальный мир через объекты и их отношения

Пример ООП (Java)

```
class Animal {
    void makeSound() {
        System.out.println("Some sound");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Bark");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        myDog.makeSound(); // Output: Bark
    }
}
```

Декларативное программирование

Описание ЧТО нужно сделать, а не КАК это сделать

Контраст с императивным программированием:
- Императивное: Как достичь результата
- Декларативное: Что должно быть результатом

Основные виды:
- Функциональное программирование
- Логическое программирование
- Программирование ограничениями

Декларативный подход часто более лаконичен и выразителен

Функциональное программирование (ФП)

Программирование через композицию функций

Ключевые концепции:
- Чистые функции (без побочных эффектов)
- Функции первого класса
- Рекурсия вместо циклов
- Неизменяемые данные
- Функции высшего порядка

Пример языков: Haskell, Lisp, Scala, JavaScript (частично)

ФП облегчает вывод и тестирование

Пример ФП (Haskell)

```
-- Рекурсивное вычисление факториала
factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n - 1)

-- Использование функций высшего порядка
main = do
    let numbers = [1, 2, 3, 4, 5]
    let squares = map (^2) numbers
    let evens = filter even numbers
    print squares
    print evens
```

Логическое программирование

Программирование через описание фактов и правил

Ключевые концепции:
- Факты: Утверждения об отношениях
- Правила: Логические выводы из фактов
- Запросы: Вопросы к системе
- Унификация: Сопоставление с образцом

Пример языков: Prolog, Mercury

Логическое программирование хорошо подходит для задач искусственного интеллекта и экспертных систем

Пример логического программирования (Prolog)

```
/* Факты */
parent(john, jim).
parent(john, ann).
parent(jim, bob).
parent(ann, mary).

/* Правила */
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.

/* Запросы */
% grandparent(john, bob). -> true
% sibling(bob, mary). -> false
```

Событийно-ориентированное программирование

Программа реагирует на события, а не определяет последовательность

Ключевые концепции:
- Event loop (цикл событий)
- Обработчики событий (event handlers)
- Асинхронность
- Callback функции

Пример применения: GUI приложения, веб-серверы, игры

Событийно-ориентированный подход эффективен для I/O bound приложений

Пример событийно-ориентированного кода (JavaScript)

```
// Обработчик клика на кнопку
document.getElementById('myButton').addEventListener('click',
function() {
    console.log('Button clicked!');
    // Асинхронная операция
    fetch('/api/data')
        .then(response => response.json())
        .then(data => console.log(data));
});

// Event loop в Node.js
const EventEmitter = require('events');
const myEmitter = new EventEmitter();

myEmitter.on('event', () => {
    console.log('Event occurred!');
});
```

Метапрограммирование

Программы, которые создают или манипулируют другими программами

Ключевые концепции:
- Генерация кода
- Рефлексия (introspection)
- Макросы
- Шаблоны (C++)

Пример языков: Lisp, Ruby, Python, C++

Метапрограммирование увеличивает выразительность

Пример метапрограммирования (Python)

```
# Декораторы как форма метапрограммирования
def debug_decorator(func):
    def wrapper(*args, **kwargs):
        print(f"Calling {func.__name__} with args: {args}, kwargs: {kwargs}")
        result = func(*args, **kwargs)
        print(f"{func.__name__} returned: {result}")
        return result
    return wrapper

@debug_decorator
def add(a, b):
    return a + b

# Метаклассы
class Meta(type):
    def __new__(cls, name, bases, dct):
        dct['created_by_meta'] = True
        return super().__new__(cls, name, bases, dct)

class MyClass(metaclass=Meta):
    pass

print(MyClass.created_by_meta)  # Output: True
```

Сравнительная таблица парадигм

Парадигма	Основная единица	Управление состоянием
Императивная	Инструкция	Изменяемое
Процедурная	Процедура/Функция	Изменяемое
ООП	Объект	Изменяемое (часто)
Функциональная	Функция	Неизменяемое
Логическая	Логическое утверждение	Н/Д
Событийно-ориентированная	Обработчик событий	Зависит от реализации

Преимущества и недостатки

Императивное:
+ Полный контроль над выполнением
- Сложнее reasoning о коде

ООП:
+ Хорошая организация сложных систем
- Overhead, сложность наследования

Функциональное:
+ Легкое тестирование, параллелизм
- Кривая обучения, производительность

Логическое:
+ Мощно для определенных задач
- Ограниченная применимость

Гибридные подходы

Современные языки часто поддерживают multiple paradigms:

Python:
- Процедурный: функции и модули
- ООП: классы и объекты
- Функциональный: map, filter, lambda
- Аспектно-ориентированный: декораторы

JavaScript:
- Прототипно-ориентированный: объекты
- Функциональный: функции первого класса
- Событийно-ориентированный: event loop

Гибридные языки позволяют выбирать лучшую парадигму для каждой задачи

Как выбрать парадигму?

Критерии выбора:

1. **Природа задачи**:
   - Алгоритмические задачи → Императивная/Функциональная
   - Моделирование систем → ООП
   - Правила и логика → Логическая

2. **Требования к производительности**
3. **Командная экспертиза**
4. **Экосистема и библиотеки**
5. **Поддержка**

Не существует "лучшей" парадигмы - есть наиболее подходящая для конкретной задачи

Будущее парадигм программирования

Тенденции:

- Конвергенция парадигм в мультипарадигмальных языках
- Рост популярности функционального программирования
- Декларативные подходы в DevOps и Infrastructure as Code
- Парадигмы для параллельных и распределенных систем
- Влияние машинного обучения на языки программирования

Теория: Будущее за гибкими языками, позволяющими комбинировать лучшие aspects разных парадигм

Заключение

Ключевые выводы:

- Парадигмы - это разные способы мышления о проблемах
- Каждая парадигма имеет свои strengths и weaknesses
- Современная разработка часто использует гибридные подходы
- Понимание парадигм делает разработчика более универсальным

"The programmer who doesn't understand multiple paradigms is like a carpenter who only knows how to use a hammer."

Парадигмы программирования в C++

1. **Процедурная парадигма**
   - Основана на понятии процедуры или подпрограммы
   - Программа представляет собой последовательность вызовов процедур
   - Пример:

```
void calculate(int a, int b) { int result = a + b; print(result); }
```

2. **Объектно-ориентированная парадигма**
   - Основана на понятиях объектов и классов
   - Основные принципы:
     - Инкапсуляция
     - Наследование
     - Полиморфизм

   Пример:

```
class Car {
private:
          std::string model;
public:
          void setModel(std::string m) { model = m; }
          std::string getModel() { return model; } };
```

3. **Обобщенная (generic) парадигма**
   - Позволяет писать универсальный код
   - Реализуется через шаблоны

   Пример:

```
template<typename T> T add(T a, T b) {
 return a + b; }
```

4. Функциональная парадигма

• Основана на использовании функций как объектов первого класса

• Использует:

  • Чистые функции

  • Лямбды

  • Высшие порядки функций

  Пример:

```
auto add = [](int a, int b) { return a + b; };
```

5. Метапрограммирование

• Программирование на уровне компиляции

• Позволяет вычислять значения и генерировать код во время компиляции

Пример:

```
template<int N> struct Factorial {
static constexpr int value = N * Factorial<N-1>::value; };
```

Важные особенности:

- C++ является мультипарадигмальным языком
- Программист может комбинировать разные парадигмы в одном проекте
- Выбор парадигмы зависит от:
    - Типа решаемой задачи
    - Требований к производительности
    - Удобства поддержки кода

Стили программирования vs Парадигмы

Стили программирования — это скорее набор рекомендаций и практик по написанию кода:

- Оформление кода
- Именование переменных
- Структура проектов
- Комментарии

Парадигмы определяют фундаментальный подход к решению задач и организации кода