УДК 004.85

Макаров Д.А.
студент
факультет «Информатика и системы управления»
Московский государственный технический университет имени Н.Э. Баумана
Россия, г. Москва
Шибанова А.Д.
студент
факультет «Робототехника и комплексная автоматизация»
Московский государственный технический университет имени Н.Э. Баумана
Россия, г. Москва

ФУНКЦИОНАЛЬНАЯ ПАРАДИГМА ПРОГРАММИРОВАНИЯ

Аннотация: в данной статье рассмотрена парадигма функционального программирования. Проведен анализ концепций функционального программирования, выделены их особенности. Указана область применения каждого из них. Описаны плюсы и минусы применения данной парадигмы.

*Ключевые слова: функция, программирование, лямбда-исчисление, рекурсия, ссылочная прозрачность***.**

Makarov D.A.
student
Faculty of Informatics and Management Systems
Moscow State Technical University named after N.E. Bauman
Russia, Moscow
Shibanova A.D.

student

Faculty of Robotics and complex automation

Moscow State Technical University named after N.E. Bauman

Russia, Moscow

FUNCTIONAL PROGRAMMING PARADIGM

Abstract: in this article, the paradigm of functional programming is considered. The analysis of the concepts of functional programming is carried out, their features are highlighted. The scope of each of them is indicated. The pros and cons of application of this paradigm are described.

Keywords: function, programming, lambda calculus, recursion, referential transparency.

Функциональное программирование - это способ думать о конструировании программного обеспечения путем создания чистых функций. Это позволяет избежать концепций общего состояния, изменчивых данных, наблюдаемых в объектно-ориентированном программировании. [1]

Функциональные языки опираются на выражения и объявления, а не на выполнение операторов. Следовательно, в отличие от других процедур, которые зависят от локального или глобального состояния, вывод значения в ФП зависит только от аргументов, переданных функции.

Характеристики функционального программирования:

- Метод функционального программирования ориентирован на результат, а не на процесс
- Акцент на том, что должно быть вычислено
- Данные неизменны

- Оно построено на концепции математических функций, которые используют условные выражения и рекурсию для выполнения вычислений
- Оно не поддерживает итерацию, такую как операторы цикла, и условные операторы, такие как If-Else. [2]

Целью любого языка ФП является имитация математических функций. Однако основной процесс вычислений отличается в функциональном программировании.

Вот некоторые наиболее известные языки функционального программирования: Haskell, SML, Clojure, Scala, Erlang, F#.

Функциональные программы должны выполнять операции так же, как в первый раз. Таким образом, вы будете знать, что могло или не могло произойти во время выполнения программы, и ее побочные эффекты. В терминах ФП это называется ссылочной прозрачностью. [3]

Модульная конструкция повышает производительность. Небольшие модули могут быть быстро закодированы и имеют больше шансов на повторное использование, что, несомненно, приводит к более быстрой разработке программ. Кроме того, модули могут тестироваться отдельно, что помогает сократить время, затрачиваемое на модульное тестирование и отладку.

Поддерживаемость - это простой термин, означающий, что программирование на ФП легче поддерживать, поскольку вам не нужно беспокоиться о случайном изменении чего-либо за пределами данной функции.

«Первоклассная функция» - это определение, приписываемое сущностям языка программирования, которые не имеют ограничений на их ис-

пользование. Поэтому первоклассные функции могут появляться в любом месте программы.

Закрытие - это внутренняя функция, которая может обращаться к переменным родительской функции даже после выполнения родительской функции.

Функции высшего порядка либо принимают другие функции в качестве аргументов, либо возвращают их в качестве результатов. [4]

Функции высшего порядка допускают частичное применение. Этот метод применяет функцию к своим аргументам по одному, поскольку каждое приложение возвращает новую функцию, которая принимает следующий аргумент.

Совместное использование состояний является важной концепцией в программировании ООП. По сути, это добавление свойств к объектам.

Побочные эффекты - это любые изменения состояния, которые происходят вне вызываемой функции. Самая большая цель любого языка программирования ФП - минимизировать побочные эффекты, отделяя их от остального программного кода. В программировании на ФП жизненно важно убрать побочные эффекты от остальной логики программирования.

Преимущества функционального программирования:

- Позволяет избежать запутанных проблем и ошибок в коде
- Проще протестировать и выполнить модульное тестирование и отладку кода ФП.
- Параллельная обработка и параллелизм
- Горячее развертывание кода и отказоустойчивость
- Обеспечивает лучшую модульность благодаря более короткому коду

- Увеличение производительности разработчика
- Поддерживает вложенные функции
- Позволяет эффективно использовать лямбда-исчисление

Ограничения функционального программирования:

- Парадигма функционального программирования непроста, поэтому ее трудно понять новичку
- Повторное использование очень сложно и требует постоянного рефакторинга

Использованные источники:

1. Роганова, Н.А. Функциональное программирование - М.: МГИУ, 2007. - 220 с.

2. Сергиевский, Г.М. Функциональное и логическое программирование: учебное пособие - М.: Academia, 2018. - 158 с.

3. Сошников, Д.В. Функциональное программирование на F# - М.: ДМК, 2011. - 192 с.

4. Хостманн, К. Функциональное программирование. SCALA для нетерпеливых - М.: ДМК, 2015. - 408 с.