Функциональное и логическое программирование

На базе языка Scala

Содержание

Список литературы

- **Одерски** М., Спун Л., Веннерс Б.. Scala. Профессиональное программирование. – СПб.: Питер, 2017.
- Wampler D., Payne A., Programming Scala. – 2 edition. – O'Reilly, 2015.
- **Хорстманн** К. Scala для нетерпеливых. – Москва: ДМК Пресс, 2013. – 408 с.
- https://www.scala-lang.org/ - Самые последние выпуски Scala, ссылки на документацию и ресурсы сообщества можно найти на сайте Scala

Раздел 1. Функциональное программирование

Тема 1.1 Декларативное программирование

Scala

Машина фон Неймана

В 1946 году Фон Нейман (с соавторами) описал в техническом докладе конкретную ЭВМ, обладающую рядом новых особенностей.

Эти новые свойства вычислительных машин, по сути, описывают архитектуру некоторого абстрактного универсального вычислителя, который сейчас принято называть **машиной Фон Неймана**. Эта машина является абстрактной моделью ЭВМ, однако, эта абстракция отличается от абстрактных исполнителей алгоритмов (например, от хорошо известной машины Тьюринга).

Машина Тьюринга может обрабатывать входные данные любого объема, поэтому этот исполнитель алгоритма принципиально нельзя реализовать.

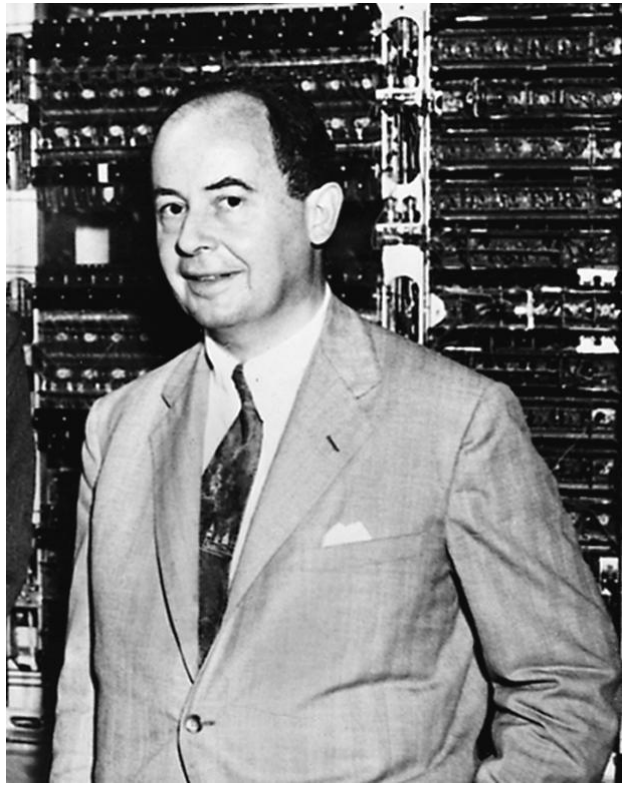


Рисунок 1 – Джон фон Нейман перед компьютером IAS

Машина фон Неймана

Машина Фон Неймана не поддаётся реализации по другой причине: многие детали в архитектуре этого вычислителя, как он описывается в учебниках, не конкретизированы.

Это сделано специально, чтобы не сковывать творческого подхода к делу у инженеров-разработчиков новых ЭВМ. Можно сказать, что машина Фон Неймана рассматривается не на внутреннем, а только на концептуальном уровне видения архитектуры.

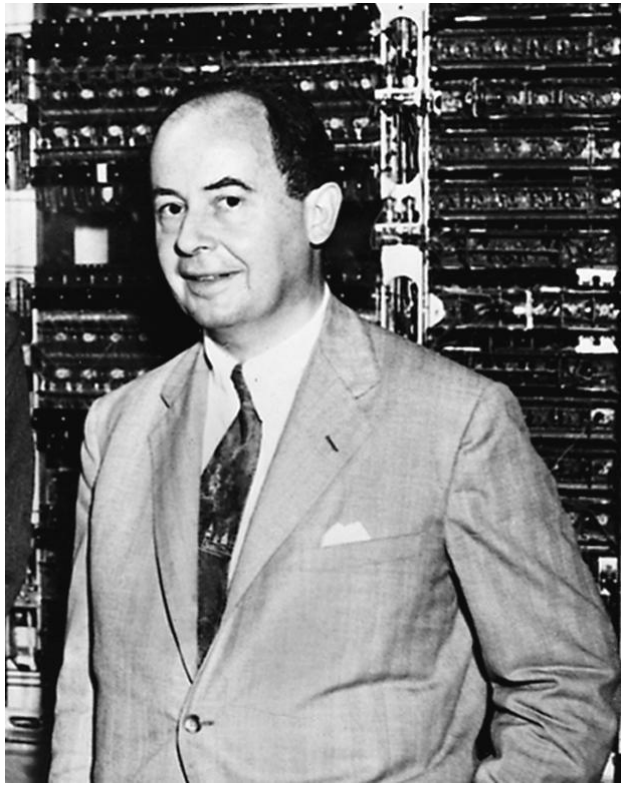


Рисунок 1 – Джон фон Нейман перед компьютером IAS

Машина фон Неймана

На схеме машины фон Неймана толстыми (двойными) стрелками показаны потоки команд и данных, а тонкими – передача между отдельными устройствами компьютера управляющих и информационных сигналов.

Выполнение каждой команды приводит к выработке последовательности управляющих сигналов, которые заставляют узлы компьютера совершать те или иные действия. С помощью же информационных сигналов одни узлы компьютера сообщают другим узлам о том, что они успешно выполнили действия, предписанные управляющими сигналами, либо зафиксировали ошибки в своей работе.

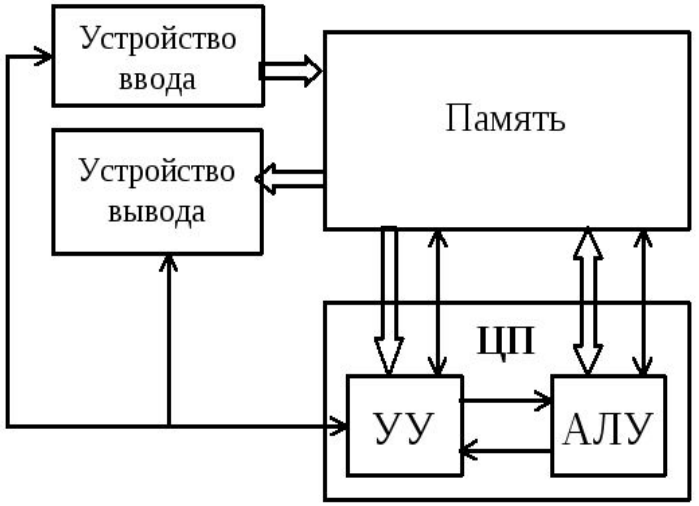


Рисунок 2 - Схема машины фон Неймана

Scala

Принципы фон Неймана

- *Принцип однородности памяти:* Команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы.
- *Принцип адресности:* Структурно основная память состоит из пронумерованных ячеек, причём процессору в произвольный момент доступна любая ячейка.
- *Принцип программного управления:* Все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности управляющих слов – команд.

Scala

Парадигмы программирования

- **Парадигма программирования** - это совокупность идей и понятий, определяющих стиль написания компьютерных программ (***подход к программированию***). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

Своим современным значением в научно-технической области термин «***парадигма***» обязан Томасу Куну и его книге «Структура научных революций». Кун называл парадигмами *устоявшиеся системы научных взглядов, в рамках которых ведутся исследования*.

Scala

Парадигмы программирования

Термин «***парадигма программирования***» впервые применил в 1978 году Роберт Флойд в своей лекции лауреата премии Тьюринга.

Флойд отмечает, что в программировании можно наблюдать явление, подобное парадигмам Куна, но, в отличие от них, парадигмы программирования не являются взаимоисключающими:

«Если прогресс искусства программирования в целом требует постоянного изобретения и усовершенствования парадигм, то совершенствование искусства отдельного программиста требует, чтобы он расширял свой репертуар парадигм.»

Scala

Декларативное программирование

- ***Декларативное программирование*** – это парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается, **что** представляет собой проблема и ожидаемый результат.

Наиболее близким к «чисто декларативному» программированию является написание ***исполнимых спецификаций***. В этом случае программа представляет собой формальную теорию, а её выполнение является одновременно автоматическим доказательством этой теории, и характерные для императивного программирования составляющие процесса разработки в этом случае исключаются: программа проектирует и доказывает сама себя.

Scala

Императивное программирование

- ***Императивное программирование*** – это парадигма программирования, для которой характерно следующее:
  - в исходном коде программы записываются инструкции (команды);
  - инструкции должны выполняться последовательно;
  - данные, получаемые при выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями;
  - данные, полученные при выполнении инструкции, могут записываться в память.

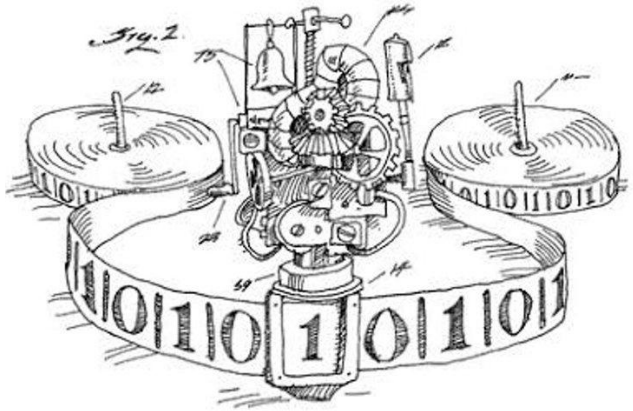
При императивном подходе к составлению кода широко используется присваивание. Наличие операторов присваивания увеличивает сложность модели вычислений и делает императивные программы подверженными специфическим ошибкам, не встречающимся при функциональном подходе.

Scala

Процедурное программирование

- ***Процедурное программирование*** – программирование на императивном языке, при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка.

Процедурное программирование является отражением архитектуры традиционных ЭВМ, которая была предложена Фон Нейманом в 1940-х годах. Теоретической моделью процедурного программирования служит абстрактная вычислительная система под названием *машина Тьюринга*.

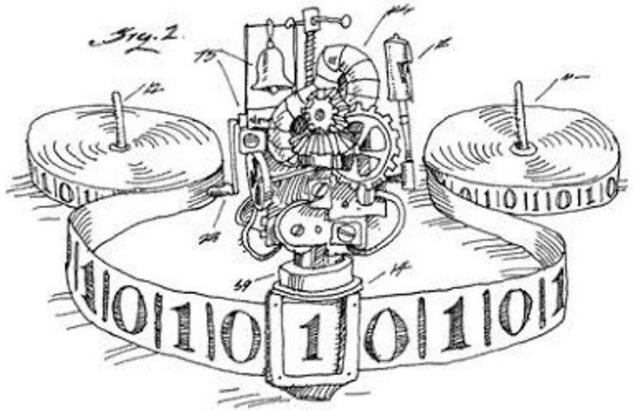


Scala

Процедурное программирование

- ***Процедурное программирование*** – программирование на императивном языке, при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка.

Процедурное программирование является отражением архитектуры традиционных ЭВМ, которая была предложена Фон Нейманом в 1940-х годах. Теоретической моделью процедурного программирования служит абстрактная вычислительная система под названием *машина Тьюринга*.



Scala

Процедурное программирование

Процедурный язык программирования предоставляет возможность программисту определять каждый шаг в процессе решения задачи. Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов.

Языки, реализующие процедурную парадигму программирования:

- Ada (язык общего назначения)
- Basic (до появления Visual Basic)
- Си
- Фортран
- Pascal
- Go

Scala

Логическое программирование

- ***Логическое программирование*** – парадигма программирования, основанная на автоматическом доказательстве теорем, а также раздел дискретной математики, изучающий принципы логического вывода информации на основе заданных фактов и правил вывода.

Логическое программирование основано на теории и аппарате математической логики с использованием математических принципов резолюций.

Scala

Структурное программирование

- ***Структурное программирование*** – парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков. Концептуализирована в конце 1960-х – начале 1970-х годов на фундаменте теоремы Бёма – Якопини, математически обосновывающей возможность структурной организации программ, и работы Эдсгера Дейкстры «О вреде оператора goto».

В соответствии с парадигмой, любая программа, которая строится без использования оператора goto, состоит из трёх базовых управляющих структур: *последовательность*, *ветвление*, *цикл*; кроме того, используются подпрограммы. При этом разработка программы ведётся пошагово, методом «сверху вниз».

Scala

Структурное программирование

Цель структурного программирования – повысить производительность труда программистов, в том числе при разработке больших и сложных программных комплексов, сократить число ошибок, упростить отладку, модификацию и сопровождение программного обеспечения.

Структурное программирование призвано, в частности, устранить беспорядок и ошибки в программах, вызванные трудностями чтения кода, несистематизированным, неудобным для восприятия и анализа исходным текстом программы. Такой текст нередко характеризуют как «***спагетти-код***».

Спагетти-код – плохо спроектированная, слабо структурированная, запутанная и трудная для понимания программа, содержащая много операторов goto (особенно переходов назад), исключений и других конструкций, ухудшающих структурированность. Самый распространённый антипаттерн программирования.

Scala

Функциональное программирование

- ***Функциональное программирование*** – раздел дискретной математики и парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм в процедурном программировании).

Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменяемость этого состояния.

Функциональное программирование

Лямбда-исчисление являются основой для функционального программирования, многие функциональные языки можно рассматривать как «надстройку» над ними.

Наиболее известными языками функционального программирования являются:

- Лисп и множество его диалектов, наиболее современные из которых:
  - Scheme
  - Clojure
  - Common Lisp
- Erlang – функциональный язык с поддержкой процессов.
  - Elixir
- Scala
- Haskell – чистый функциональный. Назван в честь Хаскелла Карри.

Scala

Функциональное программирование

Основной особенностью функционального программирования, определяющей как преимущества, так и недостатки данной парадигмы, является то, что в ней реализуется **модель вычислений без состояний.**

То, что в императивных языках делается путём присваивания значений переменным, в функциональных достигается путём передачи выражений в параметры функций. Непосредственным следствием становится то, что чисто функциональная программа не может изменять уже имеющиеся у неё данные, а может лишь порождать новые путём копирования и/или расширения старых. Следствием того же является отказ от циклов в пользу рекурсии.

Scala

Функциональное программирование

Достоинства:

1. Повышение надёжности кода
2. Удобство организации модульного тестирования
3. Возможности оптимизации при компиляции
4. Возможности параллелизма

Недостатки:

1. Отсутствие присваиваний и замена их на порождение новых данных приводят к необходимости постоянного *выделения и автоматического освобождения памяти*, поэтому в системе исполнения функциональной программы обязательным компонентом становится высокоэффективный *сборщик мусора*.
2. Нестрогая модель вычислений приводит к непредсказуемому порядку вызова функций, что создаёт проблемы при вводе-выводе.