### Table of Contents

[ApectJ 1](#_Toc12701)

[1.1. Общая характеристика и история создания: 1](#_Toc21874)

[1.2. Система типов и особенности архитектуры языка: 1](#_Toc20614)

[1.3. Примеры кода и интересные факты: 3](#_Toc4864)

[2. CoffseeScript 4](#_Toc14879)

[2.1. Общая характеристика и история создания: 4](#_Toc7183)

[2.2. Система типов и особенности архитектуры языка: 4](#_Toc18313)

[2.3. Примеры кода и интересные факты: 7](#_Toc2985)

[3. Elixir 7](#_Toc31828)

[3.1. Примеры кода и интересные факты:](#_Toc291) **[Error! Bookmark not defined.](#_Toc291)**

[3.2. Система типов и особенности архитектуры языка: 7](#_Toc1143)

[3.3. Примеры кода и интересные факты: 10](#_Toc17065)

# ApectJ

## Общая характеристика и история создания:

<<AspectJ — аспектно-ориентированное расширение языка Java, созданное компанией PARC. Язык доступен в проектах Eclipse Foundation как отдельно, так и в составе среды разработки Eclipse. Язык стал де-факто стандартом аспектно-ориентированного программирования, подчёркивая простоту и удобство для пользователей. С первого выпуска язык использует Java-подобный синтаксис.>>[[1]](#mark_AspectJ)И его выпуск был в 2001 году.

## Система типов и особенности архитектуры языка:

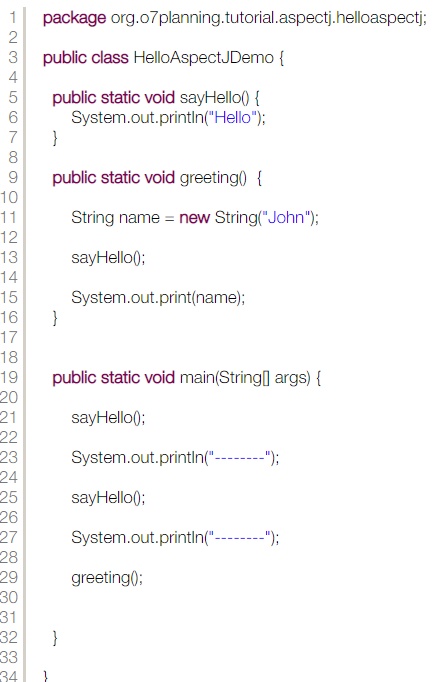
<<AspectJ расширяет синтаксис Java, то есть все программы, написанные на Java, будут корректными программами AspectJ, но не наоборот, так как могут включать специальные конструкции, называемые аспектами, которые могут содержать несколько частей, недоступных обычным классам.>>[[1]](#mark_AspectJ)Значит что системы типов JAVA и AspectJ почти одинаковые.Просто в AspectJ есть расрешение по грамматике.

AspectJ имеет понятие Join Point, где могут придать код:

|  |  |
| --- | --- |
| **Join Point** | **Значение** |
| Method call | Вызовд метода |
| Method execution | Запуск метода |
| Constructor call | Вызвод коструктора |
| Constructor execution | Запуск коструктора |
| Field get | Читать член |
| Field set | Писать член |

Table 1

## Примеры кода и интересные факты:



пример 1

[[2]](#mark_MyRu_forAs)

# CoffseeScript

## Общая характеристика и история создания:

<<CoffeeScript (кофи скрипт) — язык программирования, транслируемый в JavaScript. CoffeeScript добавляет синтаксический сахар в духе Ruby, Python, Haskell и Erlang для того, чтобы улучшить читаемость кода и уменьшить его размер. CoffeeScript позволяет писать более компактный код по сравнению с JavaScrip. JavaScript-код, получаемый трансляцией из CoffeeScript, полностью проходит проверку.>>[[3]](#mark_Coffee)

Расширение имени файлов .coffee или .litcoffee

<<Создателем языка является Джереми Ашкенас.

Изначально компилятор был написан на Ruby, но в версии 0.5, которая вышла 21 февраля 2010 года, компилятор был реализован на самом же CoffeeScript.

CoffeeScript был радушно воспринят в Ruby-сообществе. Встроенная поддержка CoffeeScript была добавлена в веб-фреймворк Ruby on Rails с версии 3.1.>>[[3]](#mark_Coffee)

## Система типов и особенности архитектуры языка:

<<Объявлять переменные в языке coffeescript не нужно. Компилятор сделает все за нас

n = 10

s = 'string'

b = true

Точки с запятой, как мы можем видеть, ставить также не надо.

Теперь о типах данных. Тут все просто, ибо они точно такие же, как и в чистом javascript: целые числа, вещественные числа, 8 и 16-ричные числа, строки, булев тип и т.д. Однако, не все осталось без изменений. Например, булев тип вы можете объявлять так:

bool = true // true

bool = yes // true

bool = on // true

bool = false // false

bool = no // false

bool = off // false

Делают то же самое, но код можно сократить.

Строки вы можете писать на нескольких строчках и не думать об экранировании переносов

string = "some very

very very long

string"

На выходе мы получиим строку без переносов, естественно. Если нам нужно, чтобы переносы сохранялись, то мы можем написать так:

string = """

Lorem ipsum dolor sit amet,

consectetur adipisicing elit.

Aliquam alias optio nostrum eligendi nihil voluptatum,

fuga labore quibusdam, provident voluptatem.

Tenetur eius voluptatum accusantium quasi quia,

dicta quidem asperiores magni.

"""

Теперь поговорим про интерполяцию строк. На примере все будет понятно

name = "Виталий"

greeting = "Привет, #{name}!"

Вот так просто мы можем на прямую в строку вставить переменную или даже выражение

name = "Виталий"

greeting = "Привет, #{name.toUpperCase()}!"

Данный метод работает только в двойных кавычках.

Перейдем к массивам.

array = [

1, 2, 3

4, 5, 6

7, 8, 9

]

Мы можем переносить все, что угодно и где угодно. Также мы можем не писать запятую после последнего в строчке элемента.

Помимо этого, в coffeescript есть диапазоны

range = [0..17] // [0, 1, 2, 3 .. 17]

range = [0...17] // [0, 1, 2, 3 ... 16]

Думаю, тут все понятно

Диапазоны также могут быть методом slice

range = [0...37]

range[0..17] // range.slice(0, 18)

range[7...17] // range.slice(7, 17)

range[..17] // range.slice(0, 18)

range[7..] // range.slice(7)

Мы также можем скопировать массив

FirstRange = [0..27]

SecondRange = FirstRange[..]

Теперь перейдем к объектным литералам

user =

name: "Виталий"

age: 17

status: "Программист"

skills:

html: 10

css: 7

javascript: 100500

coffeescript 10

Скопируйте данный код и посмотрите, что за объект получится, сами.

Теперь немного поговорим о регулярных выражениях

card = ///

^4 # Начинается с 4

[0-9]{12} # 12 цифр

(?:[0-9]{3})?$ # У новых карт 16 цифр

///

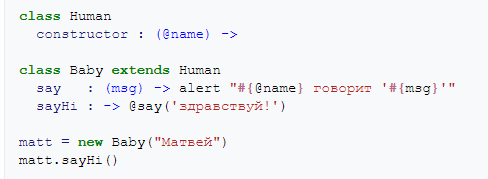
Как мы можем видеть, мы можем вставлять комментарии прямо в регулярные выражения. Однако, их, конечно же, не будет в скомпилированном javascript.>>[[4]](#mark_07_forCoffee)

Существуют разные методы, которые применяются для суждения типов:

|  |  |  |
| --- | --- | --- |
| метод | объснение | пример |
| is\_boolean | Тип boolean или нет | is\_boolean(1) //false |
| Is\_interger | Тип interger или нет | is\_integer(1) //true |
| Is\_float | Тип float или нет | is\_float(2.4)//true |
| Is\_number | Тип number или нет | is\_number(1.2) //true |
| Is\_atom | Тип atom или нет | is\_atom(:word) //true |
| Is\_binary | Тип binary или нет | is\_binary(“H”) //true |
| Is\_function | Тип function или нет | is\_function(round) //true |

Table 2

## Примеры кода и интересные факты:



пример 2

[[3]](#mark_Coffee)

# Elixir

## Общая характеристика и история создания:

<<Elixir — функциональный, распределённый язык программирования общего назначения, который работает на виртуальной машине Erlang (BEAM). Построен поверх Erlang, что обеспечивает распределённость, отказоустойчивость, исполнение в режиме мягкого реального времени, метапрограммирование с макросами и полиморфизмом, реализованным через протоколы. Использует Erlang/OTP для работы с деревьями процессов.

Создан Жозе Валимом (José Valim), ранее являвшимся одним из основных разработчиков фреймворка Ruby on Rails и сооснователем компании Plataformatec. Его целью было включить более высокую расширяемость и производительность в Erlang VM, сохраняя совместимость с инструментами и экосистемой Erlang.>>[[5]](#mark_07_forCoffee)

И его выпуск был в 2012 году.

## Система типов и особенности архитектуры языка:

<<На низком уровне Elixir использует примитивные типы, доступные в виртуальной машине Erlang. Так, список — всего лишь упорядоченный набор значений.

Elixir имеет следующие встроенные типы данных (см. также типы данных Erlang):

Целые произвольной точности

Числа с плавающей запятой

Атомы

Интервальные типы (англ. ranges)

Регулярные выражения

Идентификаторы процессов (PID) и портов

Ссылки

Кортежи

Списки

Отображения (англ. maps)

Бинарные данные (англ. binaries)

Функции

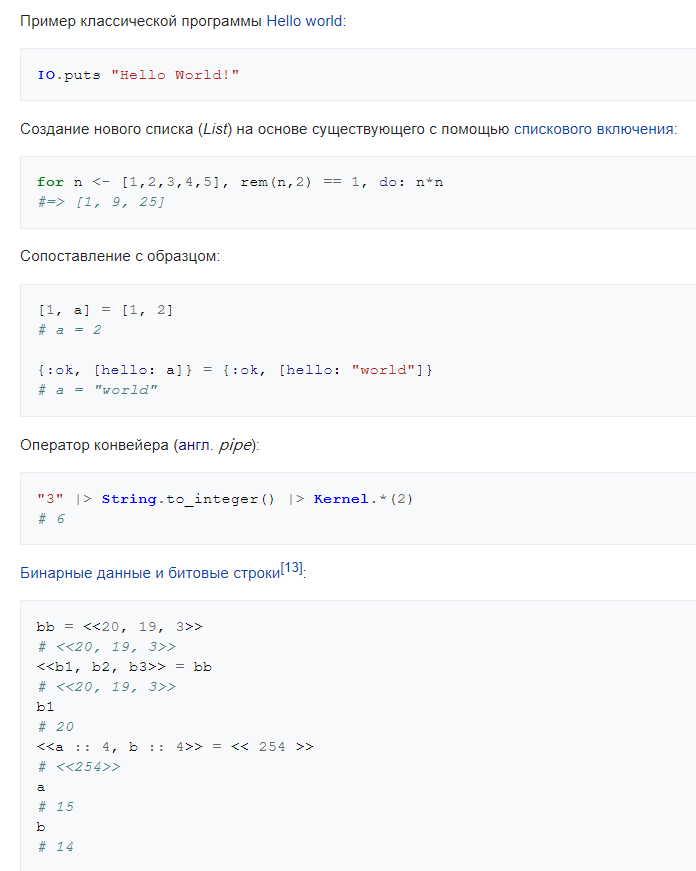
На основе этих примитивных типов в Elixir в частности построены строки и структуры.

На основе встроенных типов данных можно строить более высокоуровневые абстракции. Для построения абстракций данных в языке функционального программирования Elixir используются модули. Например, Keyword является модулем Elixir, а его реализация основана на списке кортежей, и, естественно, с Keyword можно работать и как со списком. Модули являются «чистыми» и не содержат собственного состояния как, например, объекты в языках, использующих ООП.

Например, аналогом "строка".downcase из Ruby в Elixir будет: String.downcase("строка"). Обычно экземпляр абстракции передаётся в функции модуля как первый аргумент.

Модули содержат функции, позволяющие создать абстракцию, делать запросы, а также модифицировать её (путём создания нового экземпляра). Следует заметить, что абстракции не могут содержать скрытых данных — они прозрачны для пользователей модуля. В то же время, модули могут содержать внутренние функции (определяемые с помощью defp), которые недоступны из других модулей.>>[[5]](#mark_Exlixir)

## Примеры кода и интересные факты:



пример 3

[[5]](#mark_Exlixir)

# Список использованной литературы

1. [Википедия.AspectJ](https://ru.wikipedia.org/wiki/AspectJ)
2. [MyRusakov.ru](https://myrusakov.ru/coffeescript-data-types.html)
3. [Википедия.CoffeeScript](https://ru.wikipedia.org/wiki/CoffeeScript)
4. [o7planning](https://o7planning.org/ru/10257/java-aspect-oriented-programming-tutorial-with-aspectj" \l "a27249)
5. [Википедия.Elixir](https://ru.wikipedia.org/wiki/Elixir_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))

# Опрос

1. Какой язык выше первое появление?

CoffeScript

AspectJ

Elixir

1. Какого языка появление до 2010?

A.CoffeScript B.AspectJ C.Elixir D.Все после 2010

1. Какие языки вам нравятся?

Пишите: