Java как платформа

Croc Java School

Стоит ли учить Java в 2021?

- Один из самых популярных языков
- Огромная стандартная библиотека, много сторонних
- Язык активно развивается (релизы каждые 6 месяцев)
- Сфера применения: бэкенд, чуть-чуть десктоп, Android (Kotlin наступает)
- Сообщества (JUG, конференции Joker/JPoint)

Вакансии на hh.ru (октябрь 2021):

- 1 619 Go
- 2 204 C++
- 2 306 C#
- **5 255 Java**
- 5 445 Javascript
- 6 624 Python

История Java

WORA “Write Once, Run Anywhere”

А еще:

- Простой и знакомый синтаксис (C++ like)
- Надежность и безопасность (design-time vs run-time)
- Кросс-платформенность и переносимость
- Высокая производительность

Джеймс Гослинг

1992

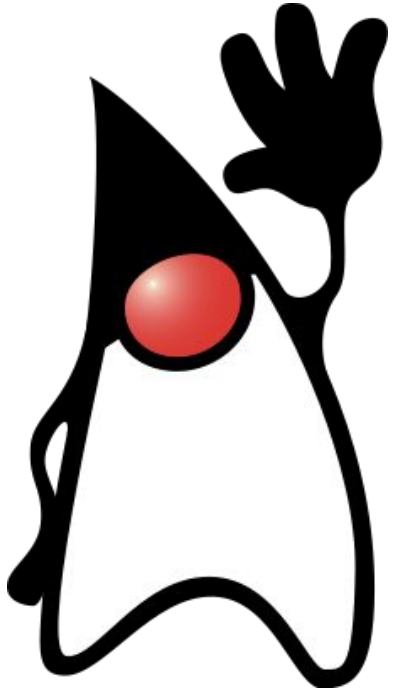
Первая демонстрация Java (устройство Star7)

https://youtu.be/1CsTH9S79gI

1996

Первая публичная версия Java 1.0


Java™



История версий

1991	Начало разработки	
1996	JDK 1.0	
2004	J2SE 5.0	*дженерики, типы перечислений, autoboxing, varargs*
2006	Java SE 6	*оптимизация производительности*
2009	Oracle покупает Sun Microsystems	
2011	Java SE 7	*оператор <>, _ в константах, строки в switch*
2014	Java SE 8	*лямбда-выражения, стримы*
2018	**Java SE 11 (LTS)**	*модульность (9), type inference (10)*
2021	Java SE 17 (LTS)	*switch expressions (14), text blocks (15), records (16)*
2023	Java SE 21 (LTS)	

В разработке

https://openjdk.java.net/projects/

- Project Valhalla: value types
- Project Loom: light threads
- Project Panama: native interop

Для реализации WORA требуется среда, гарантирующая одинаковое поведение кода на различных архитектурах, не допускающая undefined behavior (UB).

C/C++

```
int x = 2021;
int y = x / 0; // undefined behavior
```

Java

```
int x = 2021;
int y = x / 0; // ArithmeticException
```

В Java нет неопределенного поведения.

Программа на языке Java компилируется в промежуточное представление или байткод.

Виртуальная машина Java исполняет байткод на конкретной архитектуре без UB.

Особенности архитектуры:

- разрядность регистров
- набор инструкций
- порядок байтов
- ...

Референсная реализация JVM — HotSpot

http://openjdk.java.net/groups/hotspot/

Java — это остров и кофе.

Но еще:

- **язык программирования**
  - язык
  - компилятор
- **платформа**
  - виртаульная машина
  - библиотека классов

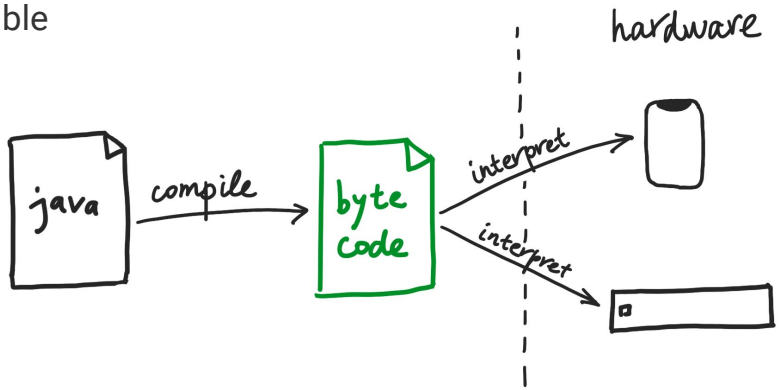
А как же JavaScript?


LINKEDIN RECRUITER
JAVA DEVELOPER
Are you interested in a position as a javascript developer?

Виртуальная машина Java

Код на языке Java компилируется в аппаратно-независимый байткод

- Байткод по отношению к коду на Java как ассемблер по отношению к коду на C/C++: строже структура, сложнее воспринимается человеком, но проще для автоматической обработки
- Не зависит от особенностей аппаратной архитектуры или ОС
- Forward-compatible



Байткод хранится в файлах .class

`0xCAFEBABE`

Так как в Java все про кофе, сигнатура формата тоже соответствует. Так начинаются все .class файлы, далее следуют 4 байта с номером версии класса.

Наборы связанных class-файлов могут быть объединены в jar-файл (архив с метаданными). Библиотеки в бинарном виде поставляются в виде jar-файлов.

Forward-compatibility

Байткод, скомпилированный старыми версиями Java, может быть исполнен на всех последующих версиях Java.

Другие JVM-языки

- Kotlin
- Groovy
- Scala
- Clojure

Чем занимается виртуальная машина?

- Загрузка и инициализация классов (байткода)
- Интерпретация байткода и JIT-компиляция
- Управление памятью, сборка мусора
- Обеспечение гарантий многопоточности (модель памяти)

Загрузка и инициализация классов (байткода)

1. Загрузка
   - Поиск ресурса (.class) и его загрузка в память виртуальной машины

2. Верификация и разрешение ссылок
   - Парсинг и проверка байткода на соответствие спецификации
   - Проверка байткода на безопасность (stack overflows, type casts, data access, etc.)
   - Выделение памяти для данных класса и разрешение ссылок

3. Инициализация
   - Установка значений полей класса по умолчанию

Интерпретация байткода и JIT-компиляция

Интерпретация

Исполнение инструкций байткода последовательно по одной.

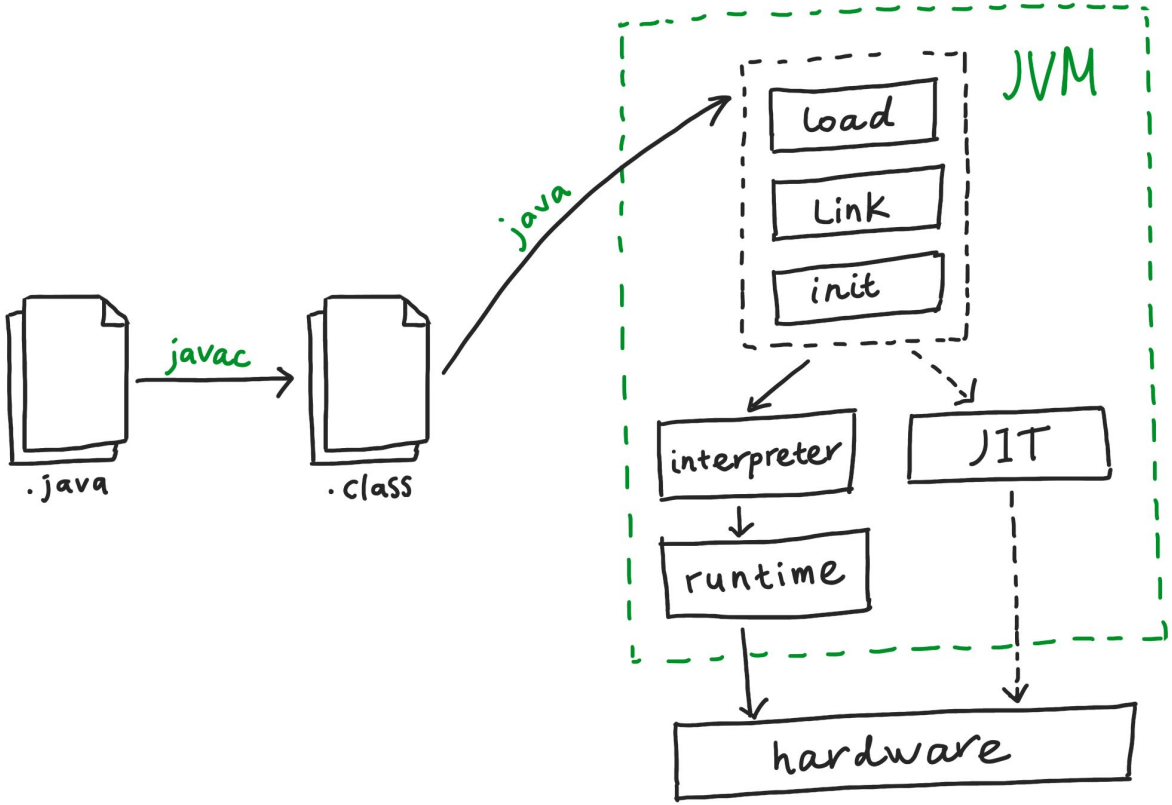
Just-in-time компиляция

Компиляция отдельных блоков байткода в нативные инструкции “на лету”

Исполнение машинного кода.

JIT vs AOT

Медленный старт (“разогрев”), но больше возможностей для оптимизации под конкретную архитектуру.


.java
javac
.class
java
JVM
load
Link
init
interpreter
JIT
runtime
hardware

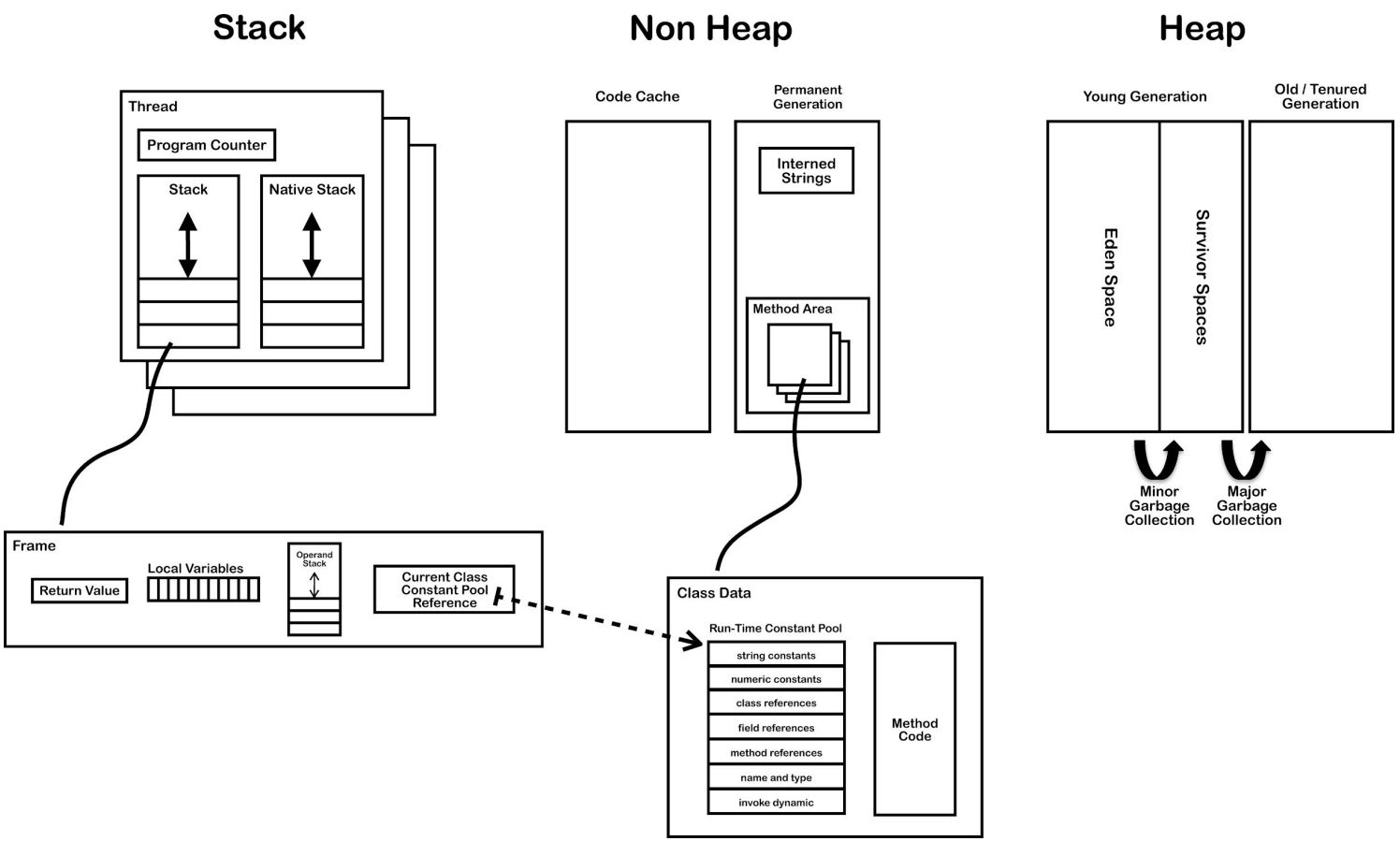
Управление памятью

Разделяемая память

- Куча (объекты, массивы)
- Non-heap memory
    - PermGen (код методов, интернированные строки)
    - Кэш (JIT)

Память потока

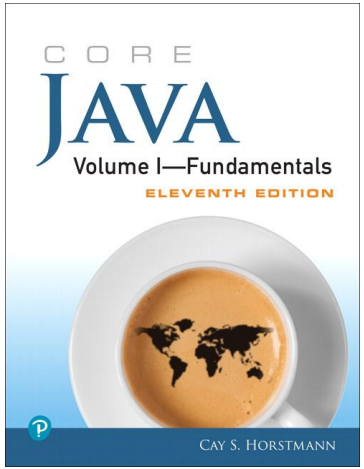
- Счетчик инструкций
- Стек (переменного или фиксированного размера)
- Нативный стек

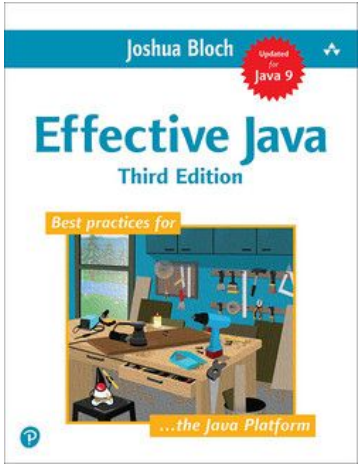


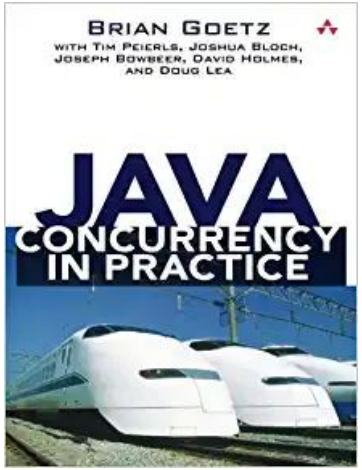
http://blog.jamesdbloom.com/JVMInternals.htm

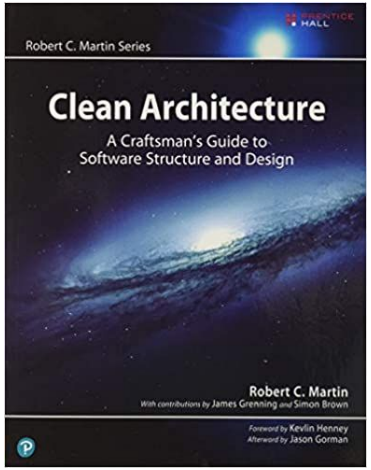
Сборка мусора

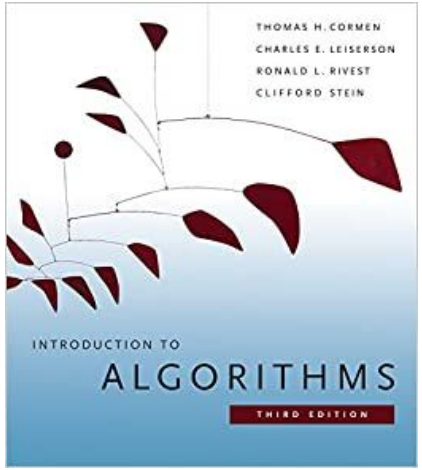
- Выполняется над данными в куче
- Удаляет объекты, на которые никто не ссылается
- STW — stop-the-world паузы неизбежны, даже если детекция объектов для удаления выполняется в отдельном потоке











Cay Horstmann. Core Java. Volume I—Fundamentals

Joshua Block. Effective Java

Brian Goetz. Java Concurrency in Practice

Robert Martin. Clean Architecture

Thomas Cormen. Introduction to Algorithms

Hello, Java ^^

```
package ru.croc.school;

public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, Java ^^");
  }
}
```

- Название класса должно совпадать с именем файла: Hello.java
- Структура директорий должна совпадать с определением package: ru/croc/school/Hello.java
- В файле может быть определен только один публичный top-level класс

Компиляция Hello.java

```
$ javac ru/croc/school/Hello.java
```

Запуск. Исполнение любой программы на Java начинается с вызова метода main()

```
$ java ru/croc/school/Hello
$ Hello, Java ^^
```

FizzBuzz

Write a short program that prints each number from 1 to 100 on a new line.

For each multiple of 3, print "Fizz" instead of the number.

For each multiple of 5, print "Buzz" instead of the number.

For numbers which are multiples of both 3 and 5, print "FizzBuzz" instead of the number.

```
public class FizzBuzz {

  public static void main(String[] args) {
    for (int n = 1; n <= 100; n++) {
      if (n % 3 == 0) {
        String str = n % 5 == 0 ? "FizzBuzz" : "Fizz";
        System.out.println(str);
      } else if (n % 5 == 0) {
        System.out.println("Buzz");
      } else {
        System.out.println(n);
      }
    }
  }
}
```