初探 GraalVM

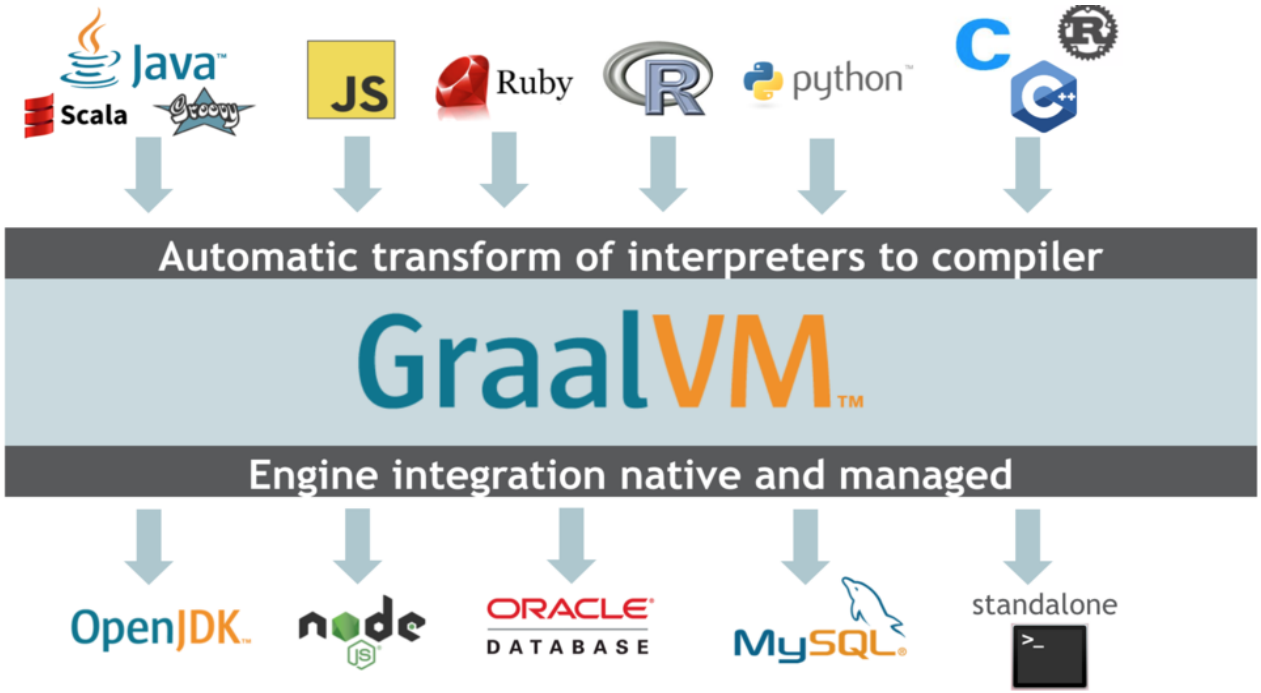
perfectpan

2019 年 9 月 6 日

GraalVM

Definition

GraalVM is a universal virtual machine for running applications written in JavaScript, Python, Ruby, R, JVM-based languages like Java, Scala, Groovy, Kotlin, Clojure, and LLVM-based languages such as C and C++.



GraalVM

- 虚拟化层代表了 GraalVM 提供的编程语言。非宿主型语言（JavaScript、Ruby、R、Python、LLVM 二进制码）能够和基于 JVM 的宿主型语言达到同样的一个运行时间，并且两者可以在同一个内存空间中来回传递数据进行互操作

GraalVM

- 虚拟化层代表了 GraalVM 提供的编程语言。非宿主型语言（JavaScript、Ruby、R、Python、LLVM 二进制码）能够和基于 JVM 的宿主型语言达到同样的一个运行时间，并且两者可以在同一个内存空间中来回传递数据进行互操作
- 消除了语言之间的界限

GraalVM

- 虚拟化层代表了 GraalVM 提供的编程语言。非宿主型语言（JavaScript、Ruby、R、Python、LLVM 二进制码）能够和基于 JVM 的宿主型语言达到同样的一个运行时间，并且两者可以在同一个内存空间中来回传递数据进行互操作
- 消除了语言之间的界限
- 提供了一个可以嵌入到另一个执行容器中的多语言执行环境，不论这个执行容器是 OpenJDK 容器还是诸如 Oracle 或者 MySQL 数据库之类的均可

构成

- Graal：动态的实时（JIT）编译器

构成

- Graal：动态的实时（JIT）编译器
- Graal Polyglot APIs

构成

- Graal：**动态的实时（JIT）编译器**
- Graal Polyglot APIs
- Graal SDK

构成

- Graal：动态的实时（JIT）编译器
- Graal Polyglot APIs
- Graal SDK
- Oracle HotSpot Java Virtual Machine (JVM): 针对那些基于 JVM 的语言或者支持非宿主编程语言提供的一个运行时环境

GraalVM 特性概览

- 代码运行的更快更高效

GraalVM 特性概览

- 代码运行的更快更高效
- 和一些更现代化的编程语言直接操作

GraalVM 特性概览

- 代码运行的更快更高效
- 和一些更现代化的编程语言直接操作
- 使得你的程序更具扩展性

Javascript & Node.js

- 你可以在 GraalVM 的环境里跑 js 和 node 程序

Javascript & Node.js

- 你可以在 GraalVM 的环境里跑 js 和 node 程序
- Execute JavaScript code with best possible performance

Javascript & Node.js

- 你可以在 GraalVM 的环境里跑 js 和 node 程序
- Execute JavaScript code with best possible performance
- 目前已支持 ES2019 语法

Javascript & Node.js

- 你可以在 GraalVM 的环境里跑 js 和 node 程序
- Execute JavaScript code with best possible performance
- 目前已支持 ES2019 语法
- 可以直接使用 Java 的类库或者框架，同时可以在 JavaScript 程序里面直接使用例如 R 或者 Python 来做数据科学计算或者绘制

Javascript & Node.js

- 你可以在 GraalVM 的环境里跑 js 和 node 程序
- Execute JavaScript code with best possible performance
- 目前已支持 ES2019 语法
- 可以直接使用 Java 的类库或者框架，同时可以在 JavaScript 程序里面直接使用例如 R 或者 Python 来做数据科学计算或者绘制
- 运行在更大的堆中

Javascript & Node.js

- 你可以在 GraalVM 的环境里跑 js 和 node 程序
- Execute JavaScript code with best possible performance
- 目前已支持 ES2019 语法
- 可以直接使用 Java 的类库或者框架，同时可以在 JavaScript 程序里面直接使用例如 R 或者 Python 来做数据科学计算或者绘制
- 运行在更大的堆中
- GraalVM 允许 C/C++ 编写的本地代码混合 JavaScript 代码

拓展之处

- Global Object: Graal

```
1 if (typeof Graal != 'undefined') {
2     print(Graal.versionJS); // GraalVM JavaScript版本号
3     print(Graal.versionGraalVM);// GraalVM的版本
4     print(Graal.isGraalRunTime);// 是不是GraalVM Javascript运行时
5 }
```

调用 Java

编译命令

js –jvm

调用 Java

编译命令

js –jvm

Java.type(className)

type 函数可以帮助导入对应的 Java 类，我们可以用 new 的方式创造实例，与 js 并无不同

调用 Java

编译命令

js –jvm

Java.type(className)

type 函数可以帮助导入对应的 Java 类，我们可以用 new 的方式创造实例，与 js 并无不同

```
1 var BigDec = Java.type('java.math.BigDecimal');
2 var bd = new BigDec("0.1");
3 console.log(bd.add(bd).toString());// 0.2
```

调用 Java

Java.to(jsData, toType)

把一个带有 length 属性的对象或数组转成 Java 的 String 或对象类型

调用 Java

Java.to(jsData, toType)

把一个带有 length 属性的对象或数组转成 Java 的 String 或对象类型

Java.isJavaObject(obj)

判断 obj 是不是 Java 的对象

Polyglot

Polyglot.export(key, value)

把 js 的任意东西导出，用 key 识别

Polyglot

Polyglot.export(key, value)

把 js 的任意东西导出，用 key 识别

```
1 function helloWorld() { print("Hello, JavaScript
      world"); }
2 Polyglot.export("helloJSWorld", helloWorld);
```

Polyglot

Polyglot.export(key, value)

把 js 的任意东西导出，用 key 识别

```
1 function helloWorld() { print("Hello, JavaScript
      world"); }
2 Polyglot.export("helloJSWorld", helloWorld);
```

Polyglot.import(key)

根据 key 值导入

Polyglot

Polyglot.eval(languageId, sourceCode)

调用指定语言的源码，返回执行完源码后的值

Polyglot

Polyglot.eval(languageId, sourceCode)

调用指定语言的源码，返回执行完源码后的值

```
1 var rArray = Polyglot.eval('R', 'runif(1000)');
2 console.log(rArray[1]); // 0.30691466969437897
```

Polyglot

Polyglot.eval(languageId, sourceCode)

调用指定语言的源码，返回执行完源码后的值

```
1 var rArray = Polyglot.eval('R', 'runif(1000)');
2 console.log(rArray[1]); // 0.30691466969437897
```

Polyglot.evalFile(languageId, sourceFileName)

相当于 import 进来一个东西

官方栗子

```
1  const express = require('express');
2  const app = express();
3  app.listen(3000);
4  app.get('/', function (req, res) {
5    var text = 'Hello GraalVM!<br/>';
6    const BigInteger = Java.type('
         java.math.BigInteger');
7    text += BigInteger.valueOf(2).pow(100).toString(16
         );
8    text += '<br/>';
9    text += Polyglot.eval('R', 'runif(100)')[0];
10   res.send(text);
11 })
```

官方栗子

Hello GraalVM!
10000000000000000000000000
0.8650917282793671

Java

- Java 如何调用 js？

Java

- Java 如何调用 js ?
- Graal SDK

Graal SDK

org.graalvm.polyglot

- 提供了 Java 调用其他语言的能力

Graal SDK

org.graalvm.polyglot

- 提供了 Java 调用其他语言的能力
- Context：给宿主语言提供了一种运行时的上下文环境

Graal SDK

org.graalvm.polyglot

- 提供了 Java 调用其他语言的能力
- Context：给宿主语言提供了一种运行时的上下文环境
- Value：代表了宿主语言的返回值

Graal SDK

Listing 1 Example.java

```
1  import org.graalvm.polyglot.*;

3  public class HelloPolyglot {
4    public static void main(String[] args) {
5      System.out.println("Hello Java!");
6      try (Context context = Context.create()) {
7        context.eval("js", "print('Hello JavaScript
             !');");
8      }
9    }
10 }
```

GraalVM SDK

Listing 2 Example2.java

```
1 try (Context context = Context.create()) {
2   Value function = context.eval("js", "x => x+1");
3   assert function.canExecute();
4   int x = function.execute(41).asInt();
5   assert x == 42;
6 }
```

GraalVM SDK

Listing 3 Example3.java

```
try (Context context = Context.create()) {
  Value result = context.eval("js",
                  "({ "                       +
                      "id   : 42, "           +
                      "text : '42', "         +
                      "arr  : [1,42,3] "      +
                  "})");
  assert result.hasMembers();

  int id = result.getMember("id").asInt();
  assert id == 42;

  String text = result.getMember("text").asString()
      ;
  assert text.equals("42");

  Value array = result.getMember("arr");
  assert array.hasArrayElements();
  assert array.getArraySize() == 3;
  assert array.getArrayElement(1).asInt() == 42;
}
```

native-image

native-image

native-image 翻译过来是本地镜像，它是基于 AOT 的编译技术，能够将 Java 代码编译成一个独立可用的软件，你可以随处调用它而不用考虑环境的问题，并且减少基于 JVM 的应用的启动内存以及加快了启动时间

native-image

native-image

native-image 翻译过来是本地镜像，它是基于 AOT 的编译技术，能够将 Java 代码编译成一个独立可用的软件，你可以随处调用它而不用考虑环境的问题，并且减少基于 JVM 的应用的启动内存以及加快了启动时间

- 比直接在 JVM 上运行相同代码要快很多

native-image

native-image

native-image 翻译过来是本地镜像，它是基于 AOT 的编译技术，能够将 Java 代码编译成一个独立可用的软件，你可以随处调用它而不用考虑环境的问题，并且减少基于 JVM 的应用的启动内存以及加快了启动时间

- 比直接在 JVM 上运行相同代码要快很多
- 但编译所需要消耗的时间和内存很大

native-image

native-image

native-image 翻译过来是本地镜像，它是基于 AOT 的编译技术，能够将 Java 代码编译成一个独立可用的软件，你可以随处调用它而不用考虑环境的问题，并且减少基于 JVM 的应用的启动内存以及加快了启动时间

- 比直接在 JVM 上运行相同代码要快很多
- 但编译所需要消耗的时间和内存很大
- 对 Java 的部分特性还不支持

AOT

定义

- 相对于 JIT

AOT

定义

- 相对于 JIT
- 在程序运行前，将字节码转成机器码，成为一个独立的运行时，拥有自己的内存管理等组件

AOT

定义

- 相对于 JIT
- 在程序运行前，将字节码转成机器码，成为一个独立的运行时，拥有自己的内存管理等组件

- GraalVM 中的 AOT 编译框架：SubstrateVM

AOT

定义

- 相对于 JIT
- 在程序运行前，将字节码转成机器码，成为一个独立的运行时，拥有自己的内存管理等组件

- GraalVM 中的 AOT 编译框架：SubstrateVM
- SubsrateVM 要求 AOT 编译的目标程序是封闭的，即不能动态加载其他类库，编译时会尽可能的探索到所有可能运行到的方法并纳入编译范围内

GodFx

技术栈

Javascript + JavaFx + GraalVM

- Javascript 开发客户端解决方案

GodFx

技术栈

Javascript + JavaFx + GraalVM

- Javascript 开发客户端解决方案
- JavaFx 编译成 native-image，暴露 API 给 js 实现交互部分

GodFx

技术栈

Javascript + JavaFx + GraalVM

- Javascript 开发客户端解决方案
- JavaFx 编译成 native-image，暴露 API 给 js 实现交互部分
- Github 地址：`https://github.com/Wooyme/GodFX`

GodFx

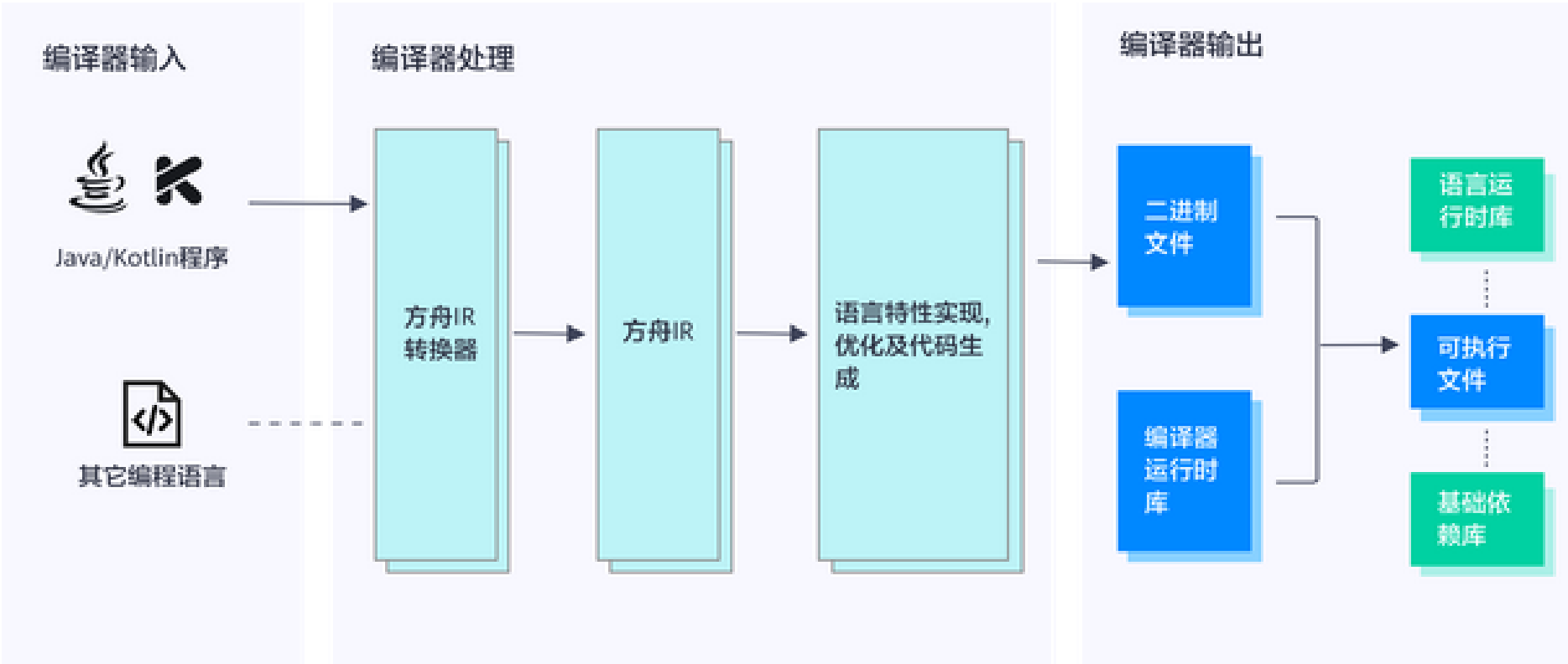
技术栈

Javascript + JavaFx + GraalVM

- Javascript 开发客户端解决方案
- JavaFx 编译成 native-image，暴露 API 给 js 实现交互部分
- Github 地址：`https://github.com/Wooyme/GodFX`
- ~~目前只有 Demo，咕咕咕~~

真假开源

- 众所周知，华为在近日" 开源" 了方舟编译器



方舟编译器架构示意图

Finally

- GraalVM 刚出来两年，还有极大的发展空间（很多的坑要踩）

Finally

- GraalVM 刚出来两年，还有极大的发展空间（很多的坑要踩）
- 混合编程的方式对前端或者后端的 Node 服务是否会带来一些不一样的改变值得思考与探索

Finally

- GraalVM 刚出来两年，还有极大的发展空间（很多的坑要踩）
- 混合编程的方式对前端或者后端的 Node 服务是否会带来一些不一样的改变值得思考与探索
- 敬请期待吧

Q&A