

## GraalVM诞生的背景

### Java在微服务/云原生时代的困境

#### 事实

有道云链接: [http://note.youdao.com/noteshare?](http://note.youdao.com/noteshare?id=f424afa1e57392b3c1fa6cc5a8ff33c2&sub=BA81092DB7FA43C081C621ADB31972D2)

[id=f424afa1e57392b3c1fa6cc5a8ff33c2&sub=BA81092DB7FA43C081C621ADB31972D2](http://note.youdao.com/noteshare?id=f424afa1e57392b3c1fa6cc5a8ff33c2&sub=BA81092DB7FA43C081C621ADB31972D2)

Java总体上是面向大规模、长时间的服务端应用而设计的。

严(luō)谨(suō)的语法利于约束所有人写出较一致的代码, 利于软件规模的提升;

但是像即时编译器(JIT)、性能优化、垃圾回收等有代表性的特征都是面向程序长时间运行设计的, 需要一段时间来达到最佳性能, 才能享受硬件规模提升带来的红利。

#### 矛盾

在微服务的背景下, 提倡服务围绕业务能力构建, 不再追求实现上的严谨一致;

1、单个微服务就不再需要再面对数十、数百GB乃至TB的内存;

2、有了高可用的服务集群, 也无须追求单个服务要7×24小时不可间断地运行, 它们随时可以中断和更新。

所以微服务对应用的容器化(Docker)亲和度(包容量、内存消耗等)、启动速度、达到最高性能的时间等方面提出了新的要求, 这些恰恰是Java的弱项。

比如: 现在启动一个微服务项目(Docker运行6个子服务), 动不动就1分钟, 如下图:

```
[61]: Started WarehouseApplication in 63.844 seconds (JVM running for 68.335)
```

#### 问题根源

##### Java离不开虚拟机

所以Java应用启动的时候, 必须要启动虚拟机, 进行类加载, 无论是启动时间, 还是占用空间都不是最优解

#### 解决方案

##### 革命派

直接革掉Java和Java生态的性命, 创造新世界, 譬如Golang

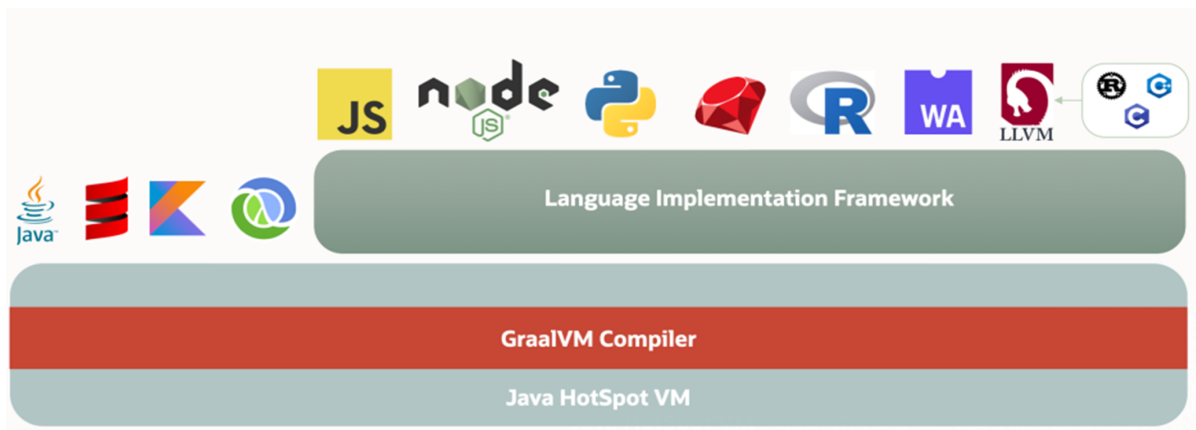
##### 保守派

尽可能保留原有主流Java生态和技术资产, 在原有的Java生态上做改进, 朝着微服务、云原生环境靠拢、适应。其中最大的技术运用就是GraalVM!

## GraalVM入门

GraalVM 是一个高性能 JDK 发行版, 旨在加速用Java和其他JVM语言编写的应用程序的执行, 并支持 JavaScript、Ruby、Python 和许多其他流行语言(翻译自官网

<https://www.graalvm.org/>)



GraalVM想成为一统天下的“最终”虚拟机！而GraalVM要做到原因也很简单：大部分脚本语言或者有动态特效的语言都需要一个语言虚拟机运行，比如CPython，Lua，Erlang，Java，Ruby，R，JS，PHP，Perl，APL等等，但是这些语言的虚拟机水平很烂，比如CPython的VM就不忍直视，而HotSpotVM是虚拟机的大神级别，如果能用上HotSpot，能用上顶级的即时编译器(JIT)、性能优化、垃圾回收等技术，岂不爽歪歪！

## GraalVM特征

**GraalVM是一款高性能的可嵌入式多语言虚拟机，它能运行不同的编程语言**

- 基于JVM的语言，比如Java, Scala, Kotlin和Groovy
- 解释型语言，比如JavaScript, Ruby, R和Python
- 配合LLVM一起工作的原生语言，比如C，C++，Rust和Swift

**GraalVM的设计目标是在不同的环境中运行程序**

- 在JVM中
- 编译成独立的本地镜像（不需要JDK环境）
- 将Java及本地代码模块集成为更大型的应用

## GraalVM下载和安装

# GraalVM分成了社区版与企业版(好消息目前都免费！)

企业版肯定比社区版好，所以推荐下载企业版，因为演示的演员，我使用的是20的版本

	GraalVM 社区	GraalVM 企业版
执照	带有“类路径”例外的 GNU 通用公共许可证 V2	用于开发/测试的Oracle 技术网络 (OTN) 许可证 ； 生产部署的商业许可证
基础 JDK	OpenJDK 11.0.13 和 17.0.1	Oracle JDK 8u311、11.0.13 和 17.0.1
支持	通过公共渠道 提供社区支持	来自Oracle 的全球 24x7 企业支持
Renaissance Suite 上的加速与 OpenJDK	1.04 倍	1.3 倍
本机映像性能与 OpenJDK (每 GB/秒的操作数)	0.82x	1.34 倍
Docker 容器映像	✓	✓
获得专利的高级编译器优化		✓

支持	通过公共渠道 提供社区支持	来自Oracle 的全球 24x7 企业支持
Renaissance Suite 上的加速与 OpenJDK	1.04 倍	1.3 倍
本机映像性能与 OpenJDK (每 GB/秒的操作数)	0.82x	1.34 倍
Docker 容器映像	✓	✓
获得专利的高级编译器优化		✓
用于低内存使用的压缩指针 (本机映像)		✓
配置文件引导优化以提高性能 (本机映像)		✓
低延迟的 G1 垃圾收集 (Native Image)		✓

<https://www.oracle.com/downloads/graalvm-downloads.html>

GaalVM Enterprise 22  
Current Release

**GraalVM Enterprise 21**  
Long-Term-Support Release

GaalVM Enterprise 20  
Long-Term-Support Release

Archived Enterprise Releases

## win10安装及配置

编辑系统变量

变量名(N):

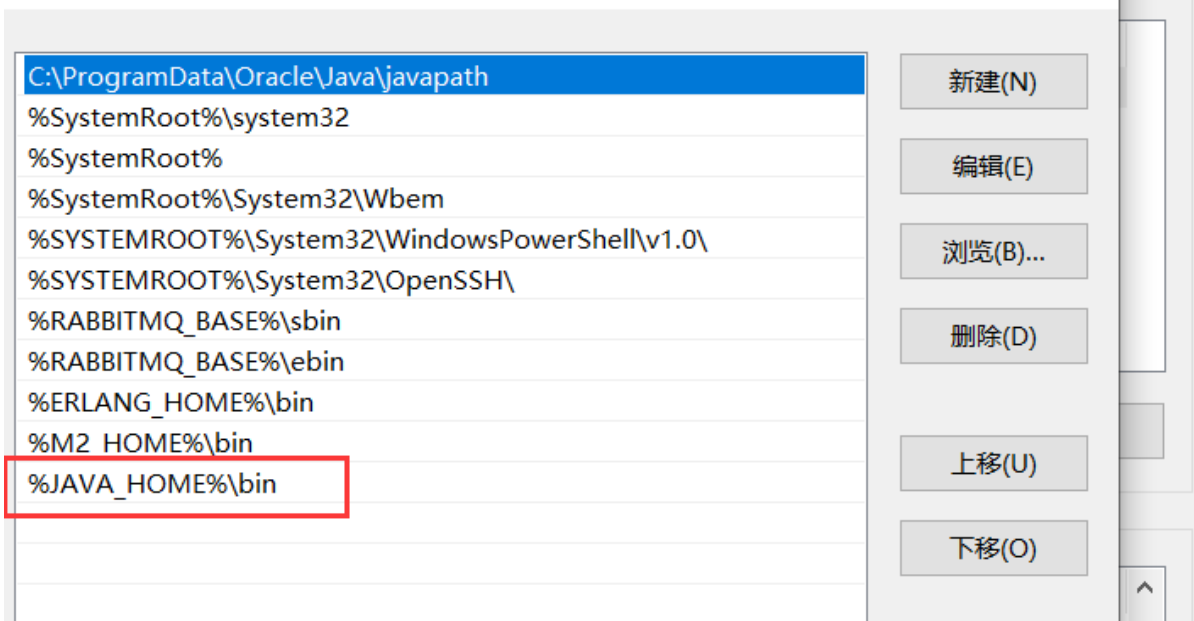
变量值(V):

浏览目录(D)...

浏览文件(F)...

确定

取消



配置时要注意，如果配置不成功，极有可能是windows10中JDK8优先了，解决方案：  
[windows10 修改java环境变量不生效](#)

```
C:\Users\Administrator\Desktop>java -version
java version "1.8.0_321"
Java(TM) SE Runtime Environment (build 1.8.0_321-07)
Java HotSpot(TM) 64-Bit Server VM GraalVM EE 21.3.1 (build 25.321-b07-jvmci-21.3-b09, mixed mode)
```

```
C:\Users\Administrator\Desktop>gu install native-image
Downloading: Release index file from oca.opensource.oracle.com
Downloading: Component catalog for GraalVM Enterprise Edition 21.3.1 on jdk8 from oca.opensource.oracle.com
Skipping ULN EE channels, no username provided.
Downloading: Component catalog from www.graalvm.org
Processing Component: Native Image
Downloading: Downloading license: Oracle GraalVM Enterprise Edition Native Image License from oca.opensource.oracle.com
The component(s) Native Image requires to accept the following license: Oracle GraalVM Enterprise Edition Native Image License
Enter "Y" to confirm and accept all the license(s). Enter "R" to the see license text.
Any other input will abort installation: Y
Downloading: Component native-image: Native Image from oca.opensource.oracle.com
Installing new component: Native Image (org.graalvm.native-image, version 21.3.1)
```

使用 GraalVM Enterprise，您可以将 Java 字节码编译为特定于平台的、自包含的本地可执行文件（本地映像 [Native Image](#)），以实现更快的启动和更小的应用程序占用空间。

安装命令如下：

```
C:\Users\Administrator\Desktop>gu install native-image
Downloading: Release index file from oca.opensource.oracle.com
Downloading: Component catalog for GraalVM Enterprise Edition 21.3.1 on jdk8 from oca.opensource.oracle.com
Skipping ULN EE channels, no username provided.
Downloading: Component catalog from www.graalvm.org
Processing Component: Native Image
Downloading: Downloading license: Oracle GraalVM Enterprise Edition Native Image License from oca.opensource.oracle.com
The component(s) Native Image requires to accept the following license: Oracle GraalVM Enterprise Edition Native Image License
Enter "Y" to confirm and accept all the license(s). Enter "R" to the see license text.
Any other input will abort installation: Y
Downloading: Component native-image: Native Image from oca.opensource.oracle.com
Installing new component: Native Image (org.graalvm.native-image, version 21.3.1)
```

```
C:\Users\Administrator\Desktop>gu list
ComponentId      Version      Component name      Stability      Origin
-----
graalvm          21.3.1      GraalVM Core        Supported
js               21.3.1      Graal.js            Supported
native-image     21.3.1      Native Image        Early adopter  oca.opensource.oracle.com
```

如果是在windows中如果要使用 本地映像（[Native Image](#)）需要安装VC，具体见：

<https://www.jianshu.com/p/a5cdf85e4ffa>

# linux安装及配置

```
[root@centosvm ~]# cd /home/king/
[root@centosvm king]# ls
3.0                                graalvm-ee-java8-21.3.1          openjdk-16+36_linux-x64_bin.tar.gz  thread.c
aot                                graalvm-ee-java8-linux-amd64-20.3.5.tar.gz  openlogic-openjdk-11.0.8+10-linux-x64.tar.gz  thread.o
graalvm-ce-java11-20.3.4          graalvm-ee-java8-linux-amd64-21.3.1.tar.gz  polyglot                                zgc
graalvm-ce-java11-linux-amd64-20.3.4.tar.gz  jdk-11.0.8                        redis-3.0
graalvm-ee-java8-20.3.5          jdk-16                            ref-jvm3.jar
```

## 解压

```
[root@centosvm king]# tar -xvf graalvm-ee-java8-linux-amd64-21.3.1.tar.gz
[root@centosvm king]# ls
3.0                                jdk-11.0.8                      ref-jvm3.jar
aot                                jdk-16                          thread.c
graalvm-ce-java11-20.3.4          openjdk-16+36_linux-x64_bin.tar.gz  thread.out
graalvm-ce-java11-linux-amd64-20.3.4.tar.gz  openlogic-openjdk-11.0.8+10-linux-x64.tar.gz  zgc
graalvm-ee-java8-21.3.1          polyglot
graalvm-ee-java8-linux-amd64-21.3.1.tar.gz  redis-3.0
```

## 配置环境变量

```
[root@centosvm king]# vi /etc/profile
```

## 修改环境变量

```
unset -f pathmunge
JAVA_HOME=/home/king/graalvm-ee-java8-20.3.5
#JAVA_HOME=/home/king/jdk-16
PATH=/home/king/graalvm-ee-java8-20.3.5/bin:$PATH
export JAVA_HOME PATH
```

```
root@centosvm king]# source /etc/profile
root@centosvm king]#
```

```
[root@centosvm ~]# java -version
java version "1.8.0_321"
Java(TM) SE Runtime Environment (build 1.8.0_321-07)
Java HotSpot(TM) 64-Bit Server VM GraalVM EE 20.3.5 (build 25.321-b07-jvmci-20.3-b28, mixed mode)
```

☐ 使用 GraalVM Enterprise，您可以将 Java 字节码编译为特定于平台的、自包含的本机可执行文件（本机映像 [Native Image](#)），以实现更快的启动和更小的应用程序占用空间。

## 安装命令如下：

```
[root@centosvm king]# gu install native-image
Downloading: Release index file from oca.opensource.oracle.com
Downloading: Component catalog for GraalVM Enterprise Edition 20.3.5 on jdk8 from oca.opensource.oracle.com
Skipping ULN EE channels, no username provided.
Downloading: Component catalog from www.graalvm.org
Processing Component: Native Image
The component(s) Native Image requires to accept the following license: Oracle GraalVM Enterprise Edition Native Image License
Enter "Y" to confirm and accept all the license(s). Enter "R" to the see license text.
Any other input will abort installation: Y
Downloading: Contents of "Oracle GraalVM Enterprise Edition Native Image License" from oca.opensource.oracle.com
Downloading: Component native-image: Native Image from oca.opensource.oracle.com
Downloading: Component native-image: Native Image from oca.opensource.oracle.com
[root@centosvm king]# gu list
ComponentId      Version      Component name      Origin
-----
js               20.3.5      Graal.js            oca.opensource.oracle.com
graalvm          20.3.5      GraalVM Core
native-image     20.3.5      Native Image
```

# GraalVM初体验(Linux)

写一个简单的类

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

编译->执行

```
[root@centosvm aot]# javac Hello.java  
[root@centosvm aot]# ls  
Hello.class Hello.java  
[root@centosvm aot]#  
[root@centosvm aot]# java Hello  
Hello World  
[root@centosvm aot]#
```

打包成一个本地可执行文件 ([Native Image](#)功能)

```
[root@centosvm aot]# native-image Hello  
[hello:4250]      classlist:    5,751.80 ms,   1.09 GB  
[hello:4250]      (cap):         884.66 ms,   1.36 GB  
[hello:4250]      setup:        5,552.63 ms,   1.36 GB  
[hello:4250]      (clinit):      262.70 ms,   1.40 GB  
[hello:4250]      (typeflow):    8,043.72 ms,   1.40 GB  
[hello:4250]      (objects):     4,061.68 ms,   1.40 GB  
[hello:4250]      (features):    205.69 ms,   1.40 GB  
[hello:4250]      analysis:    12,800.70 ms,   1.40 GB  
[hello:4250]      universe:      723.83 ms,   1.40 GB  
[hello:4250]      (parse):       3,035.90 ms,   1.40 GB  
[hello:4250]      (inline):      2,204.46 ms,   1.29 GB  
[hello:4250]      (compile):    31,252.40 ms,   1.35 GB  
[hello:4250]      compile:     37,195.90 ms,   1.35 GB  
[hello:4250]      image:         956.54 ms,   1.35 GB  
[hello:4250]      write:         260.41 ms,   1.35 GB  
[hello:4250]      [total]:     63,656.94 ms,   1.35 GB  
[root@centosvm aot]# ls  
hello Hello.class Hello.java  
[root@centosvm aot]# ls -all
```

这样就会生成一个可执行文件。这个过程我就称之为将Java 字节码编译为特定于平台的、自包含的本地可执行文件（本机映像 [Native Image](#)），以实现更快的启动和更小的应用程序占用空间。

更快速的启动：

**对比下，通过 time命令来对比**

1、通过java 走虚拟机来运行



```

[root@centosvm aot]# time java Hello
Hello World

real    0m0.118s
user    0m0.055s
sys     0m0.026s
[root@centosvm aot]#

```

2、不通过java虚拟机直接运行

```

[root@centosvm aot]# time ./hello
Hello World

real    0m0.002s
user    0m0.001s
sys     0m0.001s
[root@centosvm aot]#

```

对比发现，通过这种方式启动一个简单的类，启动速度要快很多。

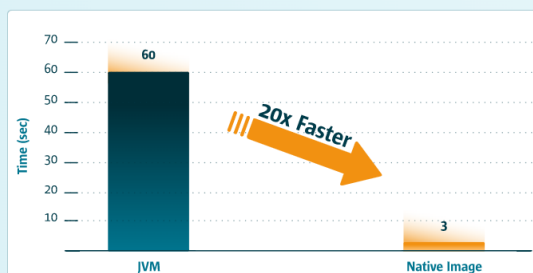
阿里早就通过这种方式加快容器的启动速度，直接启动速度提升20倍

<https://www.graalvm.org/native-image/>

## Native Image at Scale

Alibaba, a global e-commerce company, uses Native Image to compile microservices into native executables, which results in faster startup times and rapid horizontal scaling. For example, compiling for a typical microservice application which uses Spring Boot, Tomcat, and MySQL-Connector with GraalVM Native Image decreased startup time from 60 sec to just 3.

[Learn more](#)



另外这种可执行文件是不需要JDK的环境的，所以可以非常方便的完成快速的容器化部署，符合云原生的要求，例如：

我们把这个可执行文件拷贝到另外一台没有任何JDK的环境的服务器上，照样可以运行。

```

[root@centosvm king]# ls
hello jdk-11.0.8 openlogic-openjdk-11.0.8+10-linux-x64.tar.gz ref-jvm3.jar shop-order-0.0.1-SNAPSHOT
[root@centosvm king]# java -version
bash: java: 未找到命令...
[root@centosvm king]# ./hello
Hello World
[root@centosvm king]# java hello
bash: java: 未找到命令...
[root@centosvm king]# time ./hello
Hello World

real    0m0.002s
user    0m0.000s
sys     0m0.002s
[root@centosvm king]#

```

graal 的 aot 属于 “GraalVM ” 中的一项技术。

Ahead-of-time compile（提前编译），他在编译期时，会把所有相关的东西，包含一个基底的 VM，一起编译成机器码(二进制)。

好处是可以更快速的启动一个 java 应用（以往如果要启动 java 程序，需要先启动 jvm 再载入 java 代码，然后再即时的将 .class 字节码编译成机器码，交给机器执行，非常耗时间和耗内存，而如果使用AOT，可以取得一个更小更快速的镜像，适合用在云部署上）

## 多语言开发(了解即可、官网有Demo)

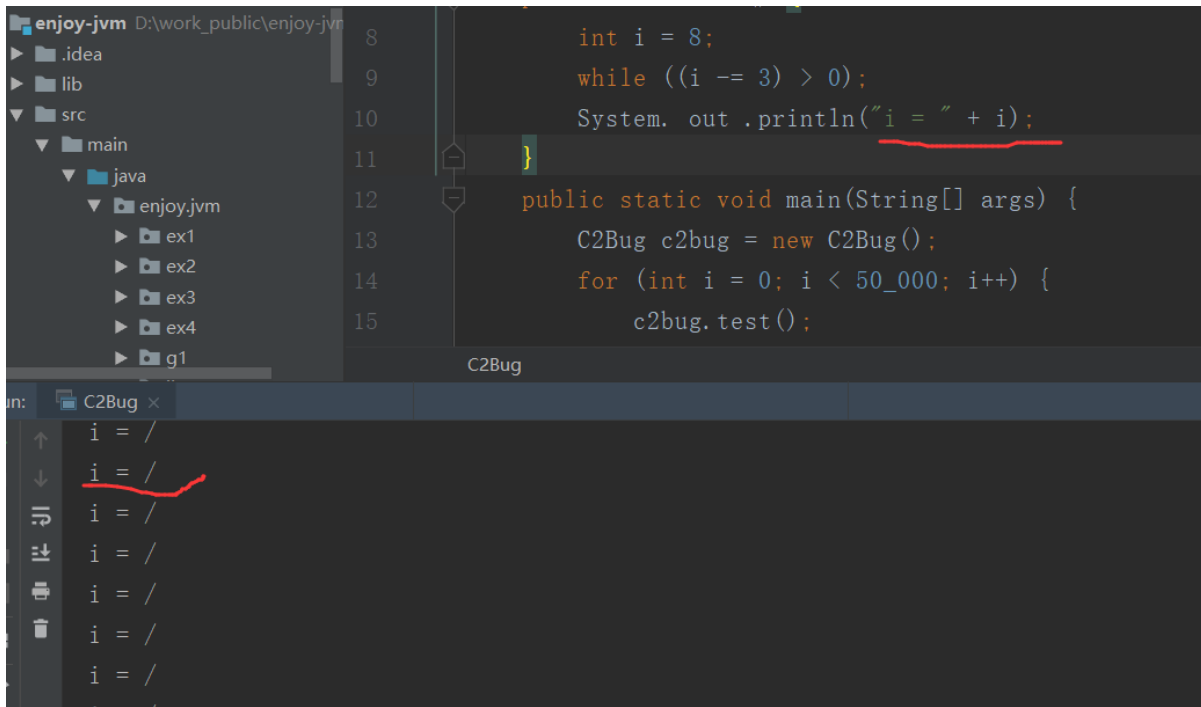
### GraalCompiler

Graal Compiler是GraalVM与HotSpotVM（从JDK10起）共同拥有的服务端即时编译器，是C2编译器的替代者。

**C2还存在一些小BUG，例如：**

```
1 public class C2Bug {
2     public void test() {
3         int i = 8;
4         while ((i -= 3) > 0);
5         System.out.println("i = " + i);
6     }
7     public static void main(String[] args) {
8         C2Bug c2bug = new C2Bug();
9         for (int i = 0; i < 50_000; i++) {
10             c2bug.test();
11         }
12     }
13 }
```



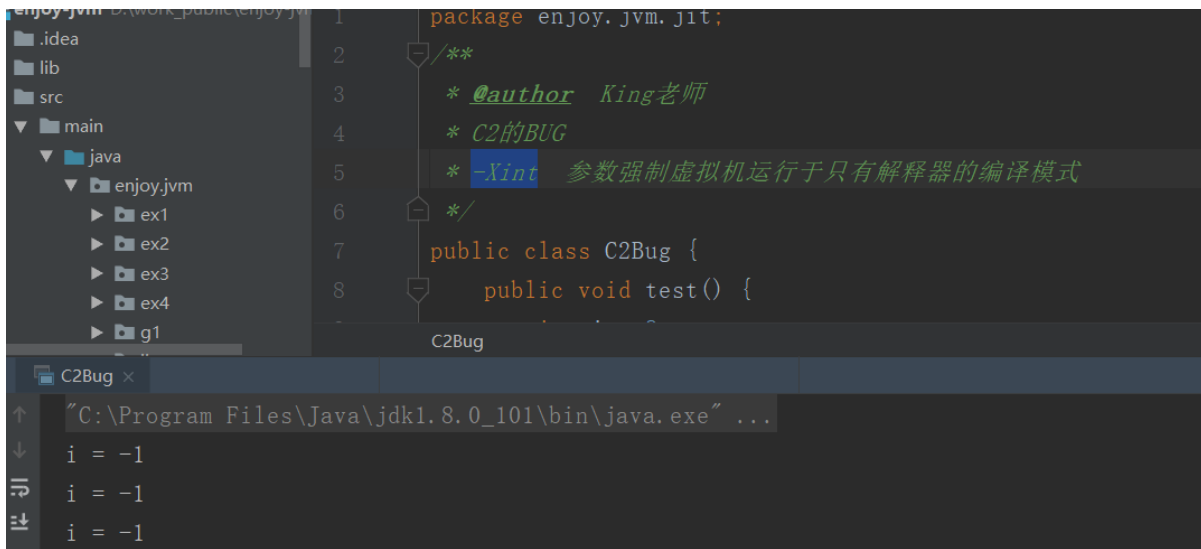


```
8      int i = 8;
9      while ((i -= 3) > 0);
10     System.out.println("i = " + i);
11 }
12
13 public static void main(String[] args) {
14     C2Bug c2bug = new C2Bug();
15     for (int i = 0; i < 50_000; i++) {
16         c2bug.test();
17     }
18 }
```

Console output:

```
i = /
i = /
i = /
i = /
i = /
i = /
i = /
```

使用-Xint 参数强制虚拟机运行于只有解释器的编译模式，就不会出现问题。



```
1 package enjoy.jvm.jit;
2
3 /**
4  * @author King老师
5  * C2的BUG
6  * -Xint 参数强制虚拟机运行于只有解释器的编译模式
7  */
8 public class C2Bug {
9     public void test() {
10         // ...
11     }
12 }
```

Console output:

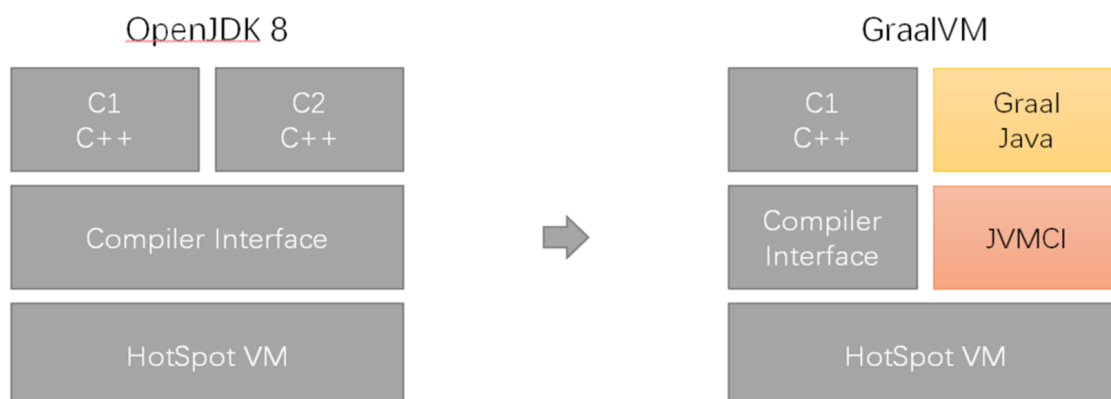
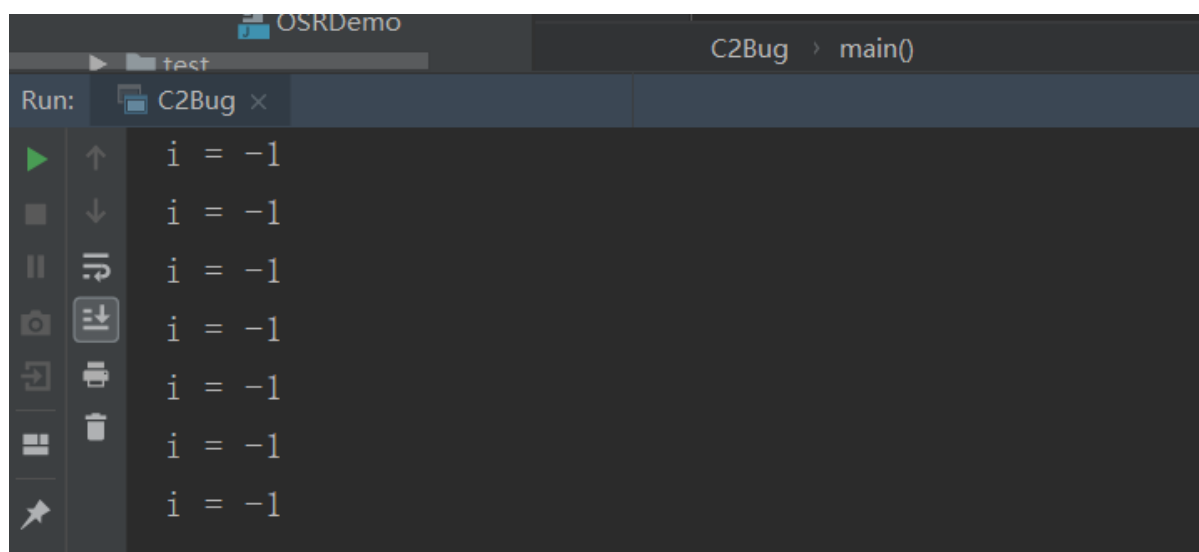
```
"C:\Program Files\Java\jdk1.8.0_101\bin\java.exe" ...
i = -1
i = -1
i = -1
```

另外把循环次数降低，降低到5000次，就不会触发JIT，就不会触发C2的优化也不会出现问题。

*\* -Xint 参数强制虚拟机运行了只解释执行的编译模式*

*\*/*

```
public class C2Bug {  
    public void test() {  
        int i = 8;  
        while ((i -= 3) > 0);  
        System.out.println("i = " + i);  
    }  
    public static void main(String[] args) {  
        C2Bug c2bug = new C2Bug();  
        for (int i = 0; i < 5_000; i++) {  
            c2bug.test();  
        }  
    }  
}
```



**即时编译器是 Java 虚拟机中相对独立的模块，它主要负责接收 Java 字节码，并生成可以直接运行的二进制码。**

传统情况下(JDK8)，即时编译器是与 Java 虚拟机紧耦合的。也就是说，对即时编译器的更改需要重新编译整个 Java 虚拟机。这对于开发相对活跃的 Graal 来说显然是不可接受的。

为了让 Java 虚拟机与 Graal 解耦合，我们引入了[Java 虚拟机编译器接口](#) (JVM Compiler Interface, JVMCI)，将即时编译器的功能抽象成一个 Java 层面的接口。这样一来，在 Graal 所依赖的 JVMCI 版本不变的情况下，我们仅需要替换 Graal 编译器相关的 jar 包 (Java 9 以后的 jmod 文件)，便可完成对 Graal 的升级

## Graal 和 C2 的区别

Graal 和 C2 最为明显的一个区别是：Graal 是用 Java 写的，而 C2 是用 C++ 写的。相对来说，Graal 更加模块化，也更容易开发与维护，毕竟，连C2的开发者都不想去维护C2了。

许多人会觉得用 C++ 写的 C2 肯定要比 Graal 快。实际上，在充分预热的情况下，Java 程序中的热点代码早已经通过即时编译转换为二进制码，在执行速度上并不亚于静态编译的 C++ 程序。

Graal 的内联算法对新语法、新语言更加友好，例如 Java 8 的 lambda 表达式以及 Scala 语言。

例如：

```
public class JavaCountUppercase {
    static final int ITERATIONS = Math.max(Integer.getInteger("iterations", 1), 1);
    public static void main(String[] args) {
        String sentence = String.join(" ", args);
        for (int iter = 0; iter < ITERATIONS; iter++) {
            if (ITERATIONS != 1) System.out.println("-- iteration " + (iter + 1) + " --");
            long total = 0, start = System.currentTimeMillis(), last = start;

            for (int i = 1; i < 100_000_000; i++) {
                total += sentence.chars().filter(Character::isUpperCase).count();
                if (i % 10_000_000 == 0) {
                    long now = System.currentTimeMillis();
                    System.out.printf("%d (%d ms)%n", i / 10_000_000, now - last);
                    last = now;
                }
            }
            System.out.printf("total: %d (%d ms)%n", total, System.currentTimeMillis() - start);
        }
    }
}
```

```
JavaCountUppercase x
"C:\Program Files\Java\jdk1.8.0_101\bin\java.exe"
1 (917 ms)
2 (796 ms)
3 (443 ms)
4 (478 ms)
5 (431 ms)
6 (481 ms)
7 (474 ms)
8 (536 ms)
9 (553 ms)
total: 0 (5585 ms)
```

**JDK8  
(C2)**

```
JavaCountUppercase x
D:\vip_tools\graalvm-ee-java-11.0.2\bin\java.exe
1 (164 ms)
2 (407 ms)
3 (19 ms)
4 (22 ms)
5 (17 ms)
6 (17 ms)
7 (17 ms)
8 (18 ms)
9 (18 ms)
total: 0 (717 ms)
```

**Graal**

## GraalVM与SpringBoot

2021年03月11日官方宣布的Spring Native只是beta版本，请不要用于生产环境！！！！

我来谈谈 GraalVM的未来发展。

spring 6.0和spring boot3.0都会基于jdk17构建，spring官方也写的很清晰，会继续维护和升级spring 2.的版本，如果有人不愿意升级，一样可以使用老的版本。

spring 6.0和spring boot3.0总体来说是彻底拥抱aot，让spring native变得更加流行，所以在Spring6与SpringBoot3广泛之前spring native还肯定不是主流而已。

同时jdk17也没有写完loom，代表着没有协程，性能方面比有协程jdk差远了。比如阿里开源的jdk8,11,就有非侵入式携程，在高并发项目中性能比jdk17要强。

所以到底是Oracle公司在引导Java了，还是有可能国内的公司引导了，这个都不好说！

**King老师在这里做一个预测：**

java的协程框架未来很大可能性是阿里等公司主导，而不是Oracle，这个是因为国内的软件行情的原因（高并发项目多）

而云原生的虚拟机则应该是Oracle公司主导，因为国外开发者，大多不会强制限定一定是Java语言，喜欢多语言开发，所以应该来说比较受欢迎，而国内则不同。