

这样做优化，实现 0.059s 启动一个SpringBoot项目！

JAVA葵花宝典 2022-10-09 18:00 发表于湖南

源：blog.csdn.net/weixin_43553153/article/details/122486769

上一篇干货：比 Redis 快 25 倍的内存数据库，杀疯了！

前言

最近自己用Spring Cloud Alibaba做了一个微服务架构的项目，部署的时候遇到了难题：内存不够。目前该项目有7个微服务，因为我只有一台阿里云的服务器(2C 4G)，所以我只能把所有的微服务部署在一台服务器上，部署方式是使用docker制作springboot的fat jar镜像，每个微服务在不加任何JVM调优参数的情况下所占内存约500M。

由于是微服务所以肯定还要部署：nacos，除此之外还用到了redis、sentinel、rocketmq、elk等(mysql买的阿里云的)，光是运行这些应用就占用内存2个多G，剩下的1个多G内存在部署4个微服务后就满了，于是开始对springboot应用的内存进行初步优化：

添加JVM参数优化内存大小

```
# JVM初始分配的内存由-Xms指定，默认是物理内存的1/64
-Xms128m
# JVM最大分配的内存由-Xmx指定，默认是物理内存的1/4
-Xmx128m
# 规定了每个线程虚拟机栈及堆栈的大小，一般情况下，256k是足够的，此配置将会影响此进程中并发
-Xss256k
# 指定并行GC线程的数量，一般最好和CPU核心数量相当
-XX:ParallelGCThreads=2
```

默认空余堆内存小于40%时，JVM就会增大堆直到 **-Xmx** 的最大限制；空余堆内存大于70%时，JVM会减少堆直到 **-Xms**的最小限制。

因此服务器一般设置 **-Xms** 、 **-Xmx** 相等以避免在每次GC 后调整堆的大小。对象的堆内存由称为垃圾回收器的自动内存管理系统回收。

默认情况下，当 CPU 数量小于8， `ParallelGCThreads` 的值等于 CPU 数量，我的服务器是2C的所以这个参数可省略。配置完成后，启动服务发现内存确实变小了，由原来的500M降至100~200M，但不是我想要的效果，我期望的效果是达到几十M的级别。

经网上查阅大量资料得知可以使用Spring Native这门新技术来实现我的需求。（该技术正处于快速迭代阶段，变动较大，建议用于个人学习，不要用于生产）

SpringBoot项目使用Spring Native后：

1. 应用启动速度特别快，毫秒级别
2. 运行时更低的内存消耗，官方展示的含有Spring Boot, Spring MVC, Jackson, Tomcat的镜像大小是50M
3. 为了达到前面的效果，代价是构建时间更长(即使是一个Hello Word构建也需要2分钟，不过主要取决于电脑配置，我的是2min左右)

Spring Native是什么

简而言之就是为了提高Java在云原生的竞争力(个人理解)。

以下内容摘抄自GitHub上Spring Native的自述文件：

Spring Native 为使用GraalVM 原生映像编译器将 Spring 应用程序编译为原生可执行文件提供 beta 支持，以提供通常设计为打包在轻量级容器中的原生部署选项。实际上，目标是在这个新平台上支持几乎未修改的 Spring Boot 应用程序。

以下内容摘抄自其他博客：

近几年“原生”一词一直泛滥在云计算、边缘计算等领域中，而原生宠幸的语言也一直都是Golang，Rust等脱离Sandbox运行的开发语言。Java得益于上世纪流行的一次编译，到处执行的理念，流行至今，但也因为这个原因，导致Java程序脱离不了JVM运行环境，使得不那么受原生程序的青睐。在云原生泛滥的今天，臃肿的JVM使Java应用程序对比其他语言显得无比的庞大，各路大神也想了很多方式让Java变的更“原生”。

Announcing Spring Native Beta!



实战

本次实战相关的环境信息如下：

- OS: Windows10 21H1
- IDE: IntelliJ IDEA 2021.2.3
- JDK: graalvm-ce-java11-21.3.0
- Maven: 3.6.3
- Docker Desktop for Windows: 20.10.12
- Spring Boot: 2.6.2
- Spring Native: 0.11.1

2. Getting Started



Applications using Spring Native should be compiled with either Java 11 or Java 17.

There are two main ways to build a Spring Boot native application:

- Using [Spring Boot Buildpacks support](#) to generate a lightweight container containing a native executable.
- Using the [GraalVM native image Maven plugin support](#) to generate a native executable.

CSDN @pcdd

从官方文档得知(上图)

使用 Spring Native 的应用程序应该使用 Java 11 或 Java 17 编译。

构建 Spring Boot 原生应用程序有两种主要方法：

1. 使用Spring Boot Buildpacks 支持生成包含本机可执行文件的轻量级容器。
2. 使用GraalVM 原生镜像 Maven 插件支持生成原生可执行文件。

经过各种踩坑后在本机上成功的使用了方法1和方法2。简单来说：

方法1就是在SpringBoot2.3后，可以使用 `spring-boot-maven-plugin` 插件来构建docker镜像，使用 `mvn spring-boot:build-image` 命令结合Docker的API来实现Spring Boot 原生应用程序的构建，成功执行后会直接生成一个docker镜像，然后run这个镜像就可以了，不用我们再写 `Dockerfile` 了，相关的参数配置都在 `pom.xml` 中配置(该插件的 `configuration` 标签下，和 `fabric8` 或 `spotify` 的 `docker-maven-plugin` 很相似)。

方法2不需要安装docker，但要安装 `Visual Studio`，然后执行 `mvn -Pnative package` 命令后会生成一个可执行文件(.exe)，运行即可。

主要区别如下

1 环境依赖不同

- 方法1需要安装Docker
- 方法2需要安装Visual Studio(需要用到部分单个组件：2个MSVC，1个Windows 10 SDK)

2 执行的maven命令不同

- 方法1是 `mvn spring-boot:build-image`
- 方法2是 `mvn -Pnative package`

因为每个微服务使用Docker部署而不是exe文件，所以方法1正好符合我的需求，所以后文使用Spring Boot Buildpacks的方式构建Spring Boot原生应用程序。

1 安装Graal VM(`graalvm-ce-java11-windows-amd64`)

官方下载地址：

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

21.3

20.3

Nightly Builds

GraalVM Community 21.3.0

- Free for all purposes
- Runs any program that runs on GraalVM Enterprise
- Based on OpenJDK 11.0.13 and 17.0.1

DOWNLOAD FROM GITHUB

DOCKER IMAGES

macOS

Linux

Windows

Details →

Release Notes →

Documentation →

GraalVM Enterprise 21.3.0

- Free for evaluation and development
- Additional performance, scalability and security
- Based on Oracle JDK 8u311, 11.0.13 and 17.0.1
- Included in Oracle Cloud and [Java SE Subscription](#)

ORACLE GRAALVM DOWNLOADS

DOCKER IMAGES

macOS

Linux

Windows

Details →

Release Notes →

Documentation →

CSDN @pcdd

Java 11 based

- Linux (amd64) - [link](#)
- Linux (aarch64) - [link](#)
- MacOS (amd64) † - [link](#)
- Windows (amd64) - [link](#)

Java 17 based

- Linux (amd64) - [link](#)
- Linux (aarch64) - [link](#)
- MacOS (amd64) † - [link](#)
- Windows (amd64) - [link](#)

† If you are using **macOS Catalina** and **later** you may need to remove the quarantine attribute from the bits before you can use them
To do this, run the following:

CSDN @pcdd

2 配置环境变量

编辑系统变量

✕

变量名(N):

GRAALVM_HOME

变量值(V):

F:\Java\graalvm-ce-java11-21.3.0

浏览目录(D)...

浏览文件(F)...

确定

取消

编辑系统变量

✕

变量名(N):

JAVA_HOME

变量值(V):

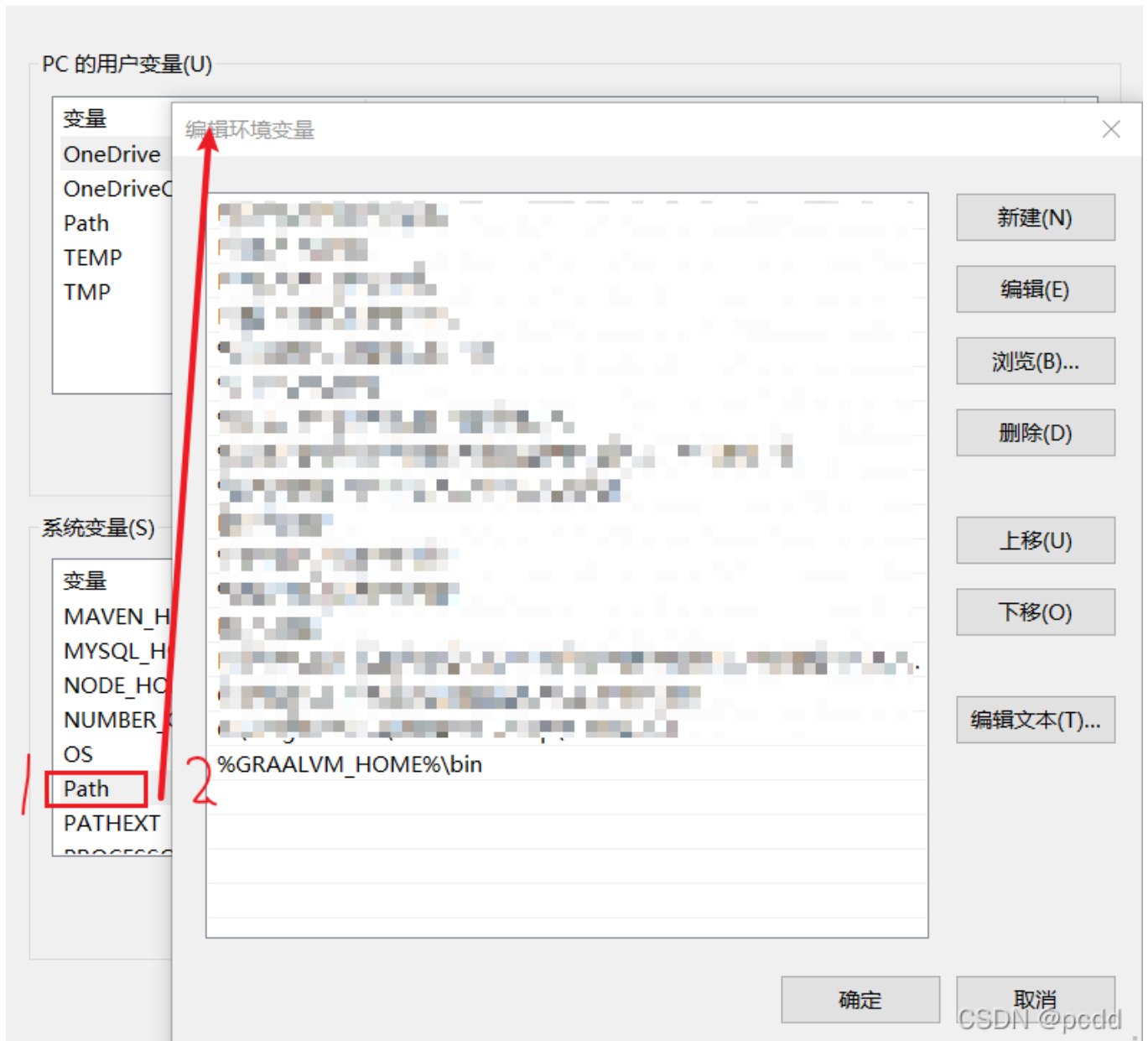
%GRAALVM_HOME%

浏览目录(D)...

浏览文件(F)...

确定

取消



针对方法1的话，上面三张图好像只用配置JAVA_HOME就行，想一次成功的话建议3个都配，后续可以自行测试。

检验是否安装成功

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS C:\Users\PC> java -version
openjdk version "11.0.13" 2021-10-19
OpenJDK Runtime Environment GraalVM CE 21.3.0 (build 11.0.13+7-jvmci-21.3-b05)
OpenJDK 64-Bit Server VM GraalVM CE 21.3.0 (build 11.0.13+7-jvmci-21.3-b05, mixed mode, sharing)
PS C:\Users\PC>
PS C:\Users\PC> mvn -v
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: F:\Apache\apache-maven-3.6.3\bin\..
Java version: 11.0.13, vendor: GraalVM Community, runtime: F:\Java\graalvm-ce-java11-21.3.0
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
PS C:\Users\PC>
```

3 安装native-image

打开新的cmd，输入以下命令，等待安装

```
gu install native-image
```

这一步我执行失败了，解决方法就是从github上手动下载 **native-image**，然后解压、安装

```
https://github.com/graalvm/graalvm-ce-builds/releases/download/vm-21.3.0/native-image-  
installable-svm-java11-windows-amd64-21.3.0.jar
```

jar用WinRAR也是可以解压的，解压后如下

A screenshot of a file explorer window showing the contents of a directory. The files listed are: bin, lib, META-INF, LICENSE_NATIVEIMAGE.txt, and native-image-installable-svm-java11-... The file LICENSE_NATIVEIMAGE.txt is highlighted with a blue background.

在bin目录下打开cmd，输入以下命令，等待安装

```
$ gu install -L native-image*
```

4 安装 Desktop for Windows

具体步骤略，按照官方文档操作即可：

```
https://docs.docker.com/desktop/windows/install/
```

5 配置pom.xml

前面都是准备工作，这一步开始才是重点

首先快速创建一个Spring Boot项目，我命名为 **spring-native**

完整的pom如下

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/m
<modelVersion>4.0.0</modelVersion>
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.6.2</version>
  <relativePath/>
</parent>
<groupId>Ltd.pcdd</groupId>
<artifactId>spring-native</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>spring-native</name>
<description>spring-native</description>
<properties>
  <java.version>11</java.version>
  <repackage.classifier/>
  <spring-native.version>0.11.1</spring-native.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.experimental</groupId>
    <artifactId>spring-native</artifactId>
    <version>${spring-native.version}</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.experimental</groupId>
      <artifactId>spring-aot-maven-plugin</artifactId>
      <version>0.11.1</version>
      <executions>
        <execution>
          <id>generate</id>
          <goals>
            <goal>generate</goal>
```



```
        </goals>
      </execution>
    </executions>
  </plugin>

  <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
      <image>
        <builder>paketobuildpacks/builder:tiny</builder>
        <env>
          <BP_NATIVE_IMAGE>true</BP_NATIVE_IMAGE>
        </env>
      </image>
    </configuration>
  </plugin>
</plugins>
</build>

<repositories>
  <repository>
    <id>spring-release</id>
    <name>Spring release</name>
    <url>https://repo.spring.io/release</url>
  </repository>
</repositories>

<pluginRepositories>
  <pluginRepository>
    <id>spring-release</id>
    <name>Spring release</name>
    <url>https://repo.spring.io/release</url>
  </pluginRepository>
</pluginRepositories>

</project>
```

本文介绍的是 **Spring Native 0.11.1** 版本，其对应的Spring Boot版本必须是2.6.2，以上只是一个最基本的配置案例，实际开发中还需要在 **spring-boot-maven-plugin** 插件的 **configuration** 标签下配置其他许许多多的参数。

例如docker远程的地址和证书的路径、jvm调优参数、配置文件指定、docker镜像名端口仓库地址等等，最好的方法就是看 **spring-boot-maven-plugin** 的官方文档，这里以配置jvm参数为例

一、简介

2. 入门

3. 使用插件

4. 目标

5.打包可执行档案

6. 打包 OCI 图像

6.1. 码头工人守护进程

6.2. Docker 注册表

6.3. 图像自定义

6.4. spring-boot:build-image

6.5. 例子

6.5.1. 自定义图像生成器

6.5.2. 生成器配置

6.5.3. 运行时 JVM 配置

6.5.4. 自定义图像名称

6.5.5. 构建包

6.5.6. 图像出版

6.5.7. 构建器缓存配置

6.5.8. 码头工人配置

6.5.3. 运行时 JVM 配置

Paketo Java buildpacks通过设置环境变量来配置 JVM 运行时环境。当应用程序映像容器中启动时，可以修改 `JAVA_TOOL_OPTIONS` buildpack-provided值以自定义 JVM 运行时行为。 `JAVA_TOOL_OPTIONS`

应该存储在映像中并应用于每个部署的环境变量修改可以按照Paketo 文档中的描述进行设置，并在以下示例中显示：

```
<project>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <image>
            <env>
              <BPE_DELIM_JAVA_TOOL_OPTIONS xml:space="preserve"> </BPE_DELIM_JAVA_TOOL_OPTIONS>
              <BPE_APPEND_JAVA_TOOL_OPTIONS>-XX:+HeapDumpOnOutOfMemoryError</BPE_APPEND_JAVA_TOOL_OPTIONS>
            </env>
          </image>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

CSDN @pcdd

通过官方文档得知只需要在 **configuration** 标签下配置即可，例如

```
<image>
<builder>paketobuildpacks/builder:tiny</builder>
<env>
  <BP_NATIVE_IMAGE>true</BP_NATIVE_IMAGE>
  <BPE_DELIM_JAVA_TOOL_OPTIONS xml:space="preserve"> </BPE_DELIM_JAVA_TOOL_OPTIONS>
  <BPE_APPEND_JAVA_TOOL_OPTIONS>-Xms128m</BPE_APPEND_JAVA_TOOL_OPTIONS>
  <BPE_APPEND_JAVA_TOOL_OPTIONS>-Xmx128m</BPE_APPEND_JAVA_TOOL_OPTIONS>
  <BPE_APPEND_JAVA_TOOL_OPTIONS>-Xss256k</BPE_APPEND_JAVA_TOOL_OPTIONS>
  <BPE_APPEND_JAVA_TOOL_OPTIONS>-XX:ParallelGCThreads=2</BPE_APPEND_JAVA_TOOL_OPTIONS>
  <BPE_APPEND_JAVA_TOOL_OPTIONS>-XX:+PrintGCDetails</BPE_APPEND_JAVA_TOOL_OPTIONS>
</env>
</image>
```

其他的配置参数还有很多。

官方文档：

<https://docs.spring.io/spring-boot/docs/2.6.2/maven-plugin/reference/htmlsingle/#build-image>

6 执行maven命令

```
mvn clean
mvn '-Dmaven.test.skip=true' spring-boot:build-image
```

下载完相关依赖后，电脑风扇就开始呼呼的转，查看任务管理器发现CPU利用率100%，内存使用量飙升，最后稳定在90%+。

构建成功

```
[INFO] [creator] Reusing cache layer 'paketo-buildpacks/bellsoft-liberica:native-image-svm'
[INFO] [creator] Reusing cache layer 'paketo-buildpacks/syft:syft'
[INFO] [creator] Adding cache layer 'paketo-buildpacks/native-image:native-image'
[INFO]
[INFO] Successfully built image 'docker.io/library/spring-native:0.0.1-SNAPSHOT'
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 05:05 min
[INFO] Finished at: 2022-01-14T18:14:46+08:00
[INFO] -----
CSDN @pcdd
```

7 创建并运行容器

查看所有镜像

```
PS C:\Users\PC> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
paketo-buildpacks/run tiny-cnb            01a3bcdeb25a       9 days ago         17.4MB
paketo-buildpacks/builder tiny                f2ab3bf3b4bd       42 years ago        509MB
spring-native       0.0.1-SNAPSHOT     cda8722111b9       42 years ago        92.1MB
```

spring-native 就是构建的镜像

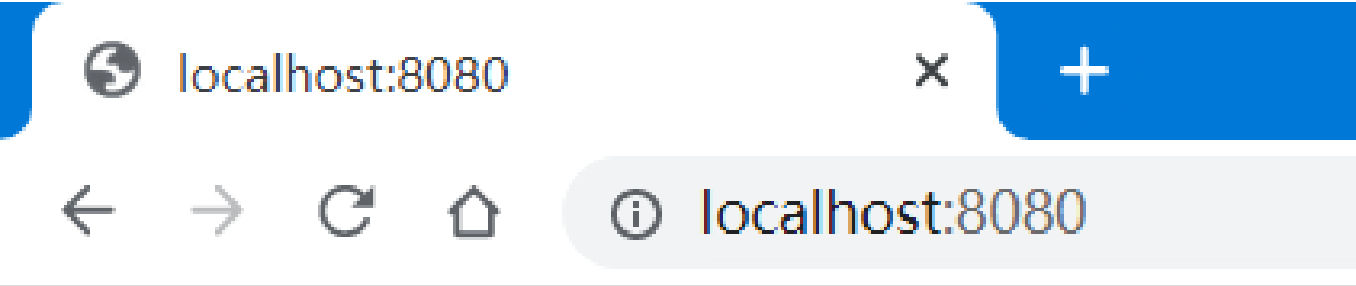
创建并运行容器

```
PS C:\Users\PC> docker run -id -p 8080:8080 --name=native-app spring-native:0.0.1-SNAPSHOT
7290b23eb151c01b9308e539a825ff92d2fc332ad93b63d47433a240766b7a88
```

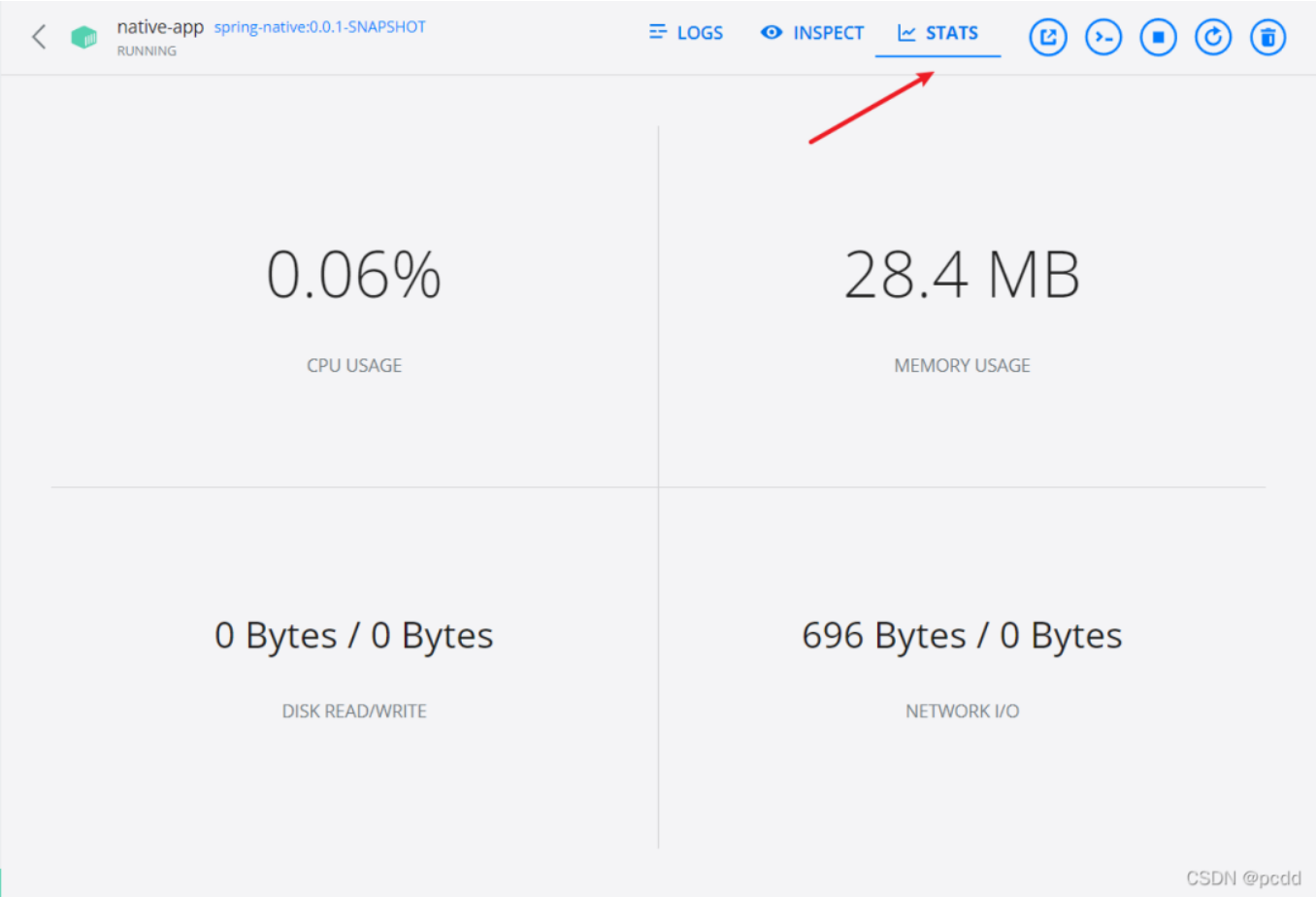
在 Docker Desktop 查看日志，发现应用成功启动，启动仅耗时。 ，也就是59ms，果然印证了 Spring Native 启动是毫秒级别这句话。

The screenshot shows the Docker Desktop interface with the 'Containers / Apps' tab selected. The 'native-app' container is running. The logs show the application starting successfully in 0.059 seconds. A red box highlights the line: 'Started SpringNativeApplication in 0.059 seconds (JVM running for 0.061)'.

成功调用接口



在 Docker Desktop 查看占用内存，仅28M左右。






不使用 Spring Native 启动应用

```
PS E:\MyCode\SpringBoot\spring-native\target> java -jar .\spring-native-0.0.1-SNAPSHOT.jar
2022-01-14 18:31:09.133 INFO 18884 --- [main] o.s.nativex.NativeListener : AOT mode disabled

:: Spring Boot :: (v0.0.1-SNAPSHOT)

2022-01-14 18:31:09.261 INFO 18884 --- [main] l.p.s.SpringNativeApplication : Starting SpringNativeApplication v0.0.1-SNAPSHOT using Java 11.0.13 on LAPTOP-K1VW4G3V
with PID 18884 (E:\MyCode\SpringBoot\spring-native\target\spring-native-0.0.1-SNAPSHOT.jar started by PC in E:\MyCode\SpringBoot\spring-native\target)
2022-01-14 18:31:09.261 INFO 18884 --- [main] l.p.s.SpringNativeApplication : The following profiles are active: default
2022-01-14 18:31:10.908 INFO 18884 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-01-14 18:31:11.008 INFO 18884 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-01-14 18:31:11.009 INFO 18884 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.56]
2022-01-14 18:31:11.087 INFO 18884 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-01-14 18:31:11.088 INFO 18884 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1751 ms
2022-01-14 18:31:11.573 INFO 18884 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-01-14 18:31:11.588 INFO 18884 --- [main] l.p.s.SpringNativeApplication : Started SpringNativeApplication in 3.025 seconds (JVM running for 3.694s) CSDN @pcdd
```

进程		性能	应用历史记录	启动	用户	详细信息	服务
名称	状态	8% CPU	43% 内存	2% 磁盘	0% 网络	3% GPU	GPU 引擎
>  IntelliJ IDEA (4)		0%	738.8 MB	0 MB/秒	0 Mbps	0%	
	 OpenJDK Platform binary	0%	511.3 MB	0 MB/秒	0 Mbps	0%	

启动耗时3s，占用内存高达511M，高下立判。

文章仅供参考，建议结合Spring Native官方最新文档学习。

<https://docs.spring.io/spring-native/docs/current/reference/htmlsingle/index.html>



JAVA葵花宝典

点击卡片关注葵花哥，让java不再难懂。Spring Boot|微服务|分布式中间件|JVM|...
74篇原创内容

公众号

来个“分享、点赞、在看”👉