

Java JVM 运行机制及基本原理



韦庆明

认真，你就赢了

[关注他](#)

233 人赞同了该文章

将知识用文字记录下来，供以后温故知新。

JVM的基础概念

JVM的中文名称叫Java虚拟机，它是由软件技术模拟出计算机运行的一个虚拟的计算机。

JVM也充当着一个翻译官的角色，我们编写出的Java程序，是不能够被操作系统所直接识别的，这时候JVM的作用就体现出来了，它负责把我们的程序翻译给系统“听”，告诉它我们的程序需要做什么操作。

我们都知道Java的程序需要经过编译后，产生Class文件，JVM才能识别并运行它，JVM针对每个操作系统开发其对应的解释器，所以只要其操作系统有对应版本的JVM，那么这份Java编译后的代码就能够运行起来，这就是Java能一次编译，到处运行的原因。

JVM的生命周期

JVM在Java程序开始执行的时候，它才运行，程序结束的时它就停止。

一个Java程序会开启一个JVM进程，如果一台机器上运行三个程序，那么就会有三个运行中的JVM进程。

JVM中的线程分为两种：守护线程和普通线程

守护线程是JVM自己使用的线程，比如垃圾回收（GC）就是一个守护线程。

普通线程一般是Java程序的线程，只要JVM中有普通线程在执行，那么JVM就不会停止。

权限足够的话，可以调用exit()方法终止程序。

JVM的结构体系



JVM的启动过程

1、JVM的装入环境和配置

在学习这个之前，我们需要了解一件事情，就是JDK和JRE的区别。

JDK是面向开发人员使用的SDK，它提供了Java的开发环境和运行环境，JDK中包含了JRE。

JRE是Java的运行环境，是面向所有Java程序的使用者，包括开发者。

JRE = 运行环境 = JVM。

如果安装了JDK，会发现电脑中有两套JRE，一套位于/Java/jre.../下，一套位于/Java/jdk.../jre下。那么问题来了，一台机器上有两套以上JRE，谁来决定运行那一套呢？这个任务就落到java.exe身上，java.exe的任务就是找到合适的JRE来运行java程序。

java.exe按照以下的顺序来选择JRE：

- 1、自己目录下有没有JRE
- 2、父目录下有没有JRE
- 3、查询注册表：HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment\“当前JRE版本号”\JavaHome

这几步的主要核心是为了找到JVM的绝对路径。

jvm.cfg的路径为：JRE路径\lib\“CPU架构”\jvm.fig

jvm.cfg的内容大致如下：

-client KNOWN

-server KNOWN

-native ERROR

-green ERROR

KNOWN 表示存在、IGNORE 表示不存在、ALIASED_TO 表示给别的JVM去一个别名

WARN 表示不存在时找一个替代、ERROR 表示不存在抛出异常

2、装载JVM

通过第一步找到JVM的路径后，Java.exe通过LoadJavaVM来装入JVM文件。

LoadLibrary装载JVM动态连接库，然后把JVM中的到处函数JNI_CreateJavaVM和JNI_GetDefaultJavaVMIntArgs 挂接到InvocationFunction 变量的CreateJavaVM和GetDefaultJavaVMInitArgs 函数指针变量上。JVM的装载工作完成。

3、初始化JVM，获得本地调用接口

调用InvocationFunction -> CreateJavaVM也就是JVM中JNI_CreateJavaVM方法获得JNIEnv结构的实例。

4、运行Java程序

JVM运行Java程序的方式有两种：jar包 与 Class

运行jar 的时候，Java.exe调用GetMainClassName函数，该函数先获得JNIEnv实例然后调用JarFileJNIEnv类中getManifest()，从其返回的Manifest对象中取getAttributes("Main-Class")的值，即jar 包中文件：META-INF/MANIFEST.MF指定的Main-Class的主类名作为运行的主类。之后main函数会调用Java.c中LoadClass方法装载该主类（使用JNIEnv实例的FindClass）。

运行Class的时候，main函数直接调用Java.c中的LoadClass方法装载该类。

Class文件

Class文件由Java编译器生成，我们创建的Java文件在经过编译器后，会变成Class的文件，这样才能被JVM所识别并运行。

类加载子系统

类加载子系统也可以称之为类加载器，JVM默认提供三个类加载器：

1、BootStrap ClassLoader：称之为启动类加载器，是最顶层的类加载器，负责加载JDK中的核心类库，如 rt.jar、resources.jar、charsets.jar等。

2、Extension ClassLoader：称之为扩展类加载器，负责加载Java的扩展类库，默认加载\$JAVA_HOME中jre/lib/*.jar 或 -Djava.ext.dirs指定目录下的jar包。

3、App ClassLoader：称之为系统类加载器，负责加载应用程序classpath目录下所有jar和class文件。

知乎

ClassLoader 之外的另外两个默认加载器都是继承自`java.lang.ClassLoader`。 `BootStrap ClassLoader` 不是一个普通的Java类，它底层由C++编写，已嵌入到了JVM的内核当中，当JVM启动后，`BootStrap ClassLoader` 也随之启动，负责加载完核心类库后，并构造`Extension ClassLoader` 和`App ClassLoader` 类加载器。

类加载器子系统不仅仅负责定位并加载类文件，它还严格按照以下步骤做了很多事情：

- 1、加载：寻找并导入`Class`文件的二进制信息
- 2、连接：进行验证、准备和解析
 - 1) 验证：确保导入类型的正确性
 - 2) 准备：为类型分配内存并初始化为默认值
 - 3) 解析：将字符引用解析为直接引用
- 3、初始化：调用Java代码，初始化类变量为指定初始值

详细请参考另一篇文章：[Java类加载机制 - 知乎专栏](#)

方法区 (Method Area)

在JVM中，**类型信息和类静态变量**都保存在方法区中，类型信息是由类加载器在类加载的过程中从类文件中提取出来的信息。

需要注意的一点是，**常量池也存放于方法区中**。

程序中所有的线程共享一个方法区，所以访问方法区的信息必须确保线程是安全的。如果有两个线程同时去加载一个类，那么只能有一个线程被允许去加载这个类，另一个必须等待。

在程序运行时，方法区的大小是可以改变的，程序在运行时可以扩展。

方法区也可以被垃圾回收，但条件非常严苛，必须在该类没有任何引用的情况下，详情可以参考另一篇文章：[Java性能优化之JVM GC（垃圾回收机制） - 知乎专栏](#)

类型信息包括什么？

- 1、类型的全名（The fully qualified name of the type）
- 2、类型的父类型全名（除非没有父类型，或者父类型是`java.lang.Object`）（The fully qualified name of the superclass）
- 3、该类型是一个类还是接口（class or an interface）（Whether or not the type is a class）
- 4、类型的修饰符（`public`, `private`, `protected`, `static`, `final`, `volatile`, `transient`等）（The modifiers of the type）
- 5、所有父接口全名的列表（An ordered list of the fully qualified names of any direct super interfaces）
- 6、类型的字段信息（Field information）
- 7、类型的方法信息（Method information）
- 8、所有静态类变量（非常量）信息（All class (static) variables declared in the type, except for constants）
- 9、一个指向类加载器的引用（A reference to class `ClassLoader`）



方法列表 (Method Tables)

为了更高效的访问所有保存在方法区中的数据，在方法区中，除了保存上边的这些类型信息之外，还有一个为了加快存取速度而设计的数据结构：方法列表。每一个被加载的非抽象类，Java虚拟机都会为他们产生一个方法列表，这个列表中保存了这个类可能调用的所有实例方法的引用，保存那些父类中调用的方法。

Java堆 (JVM堆、Heap)

当Java创建一个类的实例对象或者数组时，都在堆中为新的对象分配内存。

虚拟机中只有一个堆，程序中所有的线程都共享它。

堆占用的内存空间是最多的。

堆的存取类型为管道类型，先进先出。

在程序运行中，可以动态的分配堆的内存大小。

堆的内存资源回收是交给JVM GC进行管理的，详情请参考：[Java性能优化之JVM GC（垃圾回收机制） - 知乎专栏](#)

Java栈 (JVM栈、Stack)

在Java栈中只保存基础数据类型（参考：[Java 基本数据类型 - 四类八种 - 知乎专栏](#)）和自定义对象的引用，注意只是对象的引用而不是对象本身哦，对象是保存在堆区中的。

拓展知识：像String、Integer、Byte、Short、Long、Character、Boolean这六个属于包装类型，它们是存放于堆中的。

栈的存取类型为类似于水杯，先进后出。

栈内的数据在超出其作用域后，会被自动释放掉，它不由JVM GC管理。

每一个线程都包含一个栈区，每个栈中的数据都是私有的，其他栈不能访问。

每个线程都会建立一个操作栈，每个栈又包含了若干个栈帧，每个栈帧对应着每个方法的每次调用，每个栈帧包含了三部分：

局部变量区（方法内基本类型变量、变量对象指针）

操作数栈区（存放方法执行过程中产生的中间结果）

本地方法栈

本地方法栈的功能和JVM栈非常类似，用于存储本地方法的局部变量表，本地方法的操作数栈等信息。

栈的存取类型为类似于水杯，先进后出。

栈内的数据在超出其作用域后，会被自动释放掉，**它不由JVM GC管理。**

每一个线程都包含一个栈区，每个栈中的数据都是私有的，其他栈不能访问。

本地方法栈是在程序调用或JVM调用**本地方法接口（Native）**时候启用。

本地方法都不是使用Java语言编写的，比如使用C语言编写的本地方法，本地方法也不由JVM去运行，所以本地方法的运行不受JVM管理。

HotSpot VM将本地方法栈和JVM栈合并了。

程序计数器

在JVM的概念模型里，字节码解释器工作时就是通过改变这个计数器的值来选取下一条需要执行的字节码指令。分支、循环、跳转、异常处理、线程恢复等基础功能都需要依赖这个计数器来完成。

JVM的多线程是通过线程轮流切换并分配处理器执行时间的方式来实现的，为了各条线程之间的切换后计数器能恢复到正确的执行位置，所以**每条线程都会有一个独立的程序计数器。**

程序计数器仅占很小的一块内存空间。

当线程正在执行一个Java方法，程序计数器记录的是正在执行的JVM字节码指令的地址。如果正在执行的是一个Native（本地方法），那么这个计数器的值则为空（Undefined）。

程序计数器这个内存区域是唯一一个在JVM规范中没有规定任何OutOfMemoryError（内存不足错误）的区域。

JVM执行引擎

Java虚拟机相当于一台虚拟的“物理机”，这两种机器都有代码执行能力，其区别主要是物理机的执行引擎是直接建立在处理器、硬件、指令集和操作系统层面上的。而JVM的执行引擎是自己实现的，因此程序员可以自行制定指令集和执行引擎的结构体系，因此能够执行那些不被硬件直接支持的指令集格式。

在JVM规范中制定了虚拟机字节码执行引擎的概念模型，这个模型称之为JVM执行引擎的统一外

输入的是字节码文件、处理过程是等效字节码解析过程、输出的是执行结果。在这三点上每个JVM执行引擎都是一致的。

本地方法接口 (JNI)

JNI是Java Native Interface的缩写，它提供了若干的API实现了Java和其他语言的通信（主要是C和C++）。

JNI的适用场景

当我们有一些旧的库，已经使用C语言编写好了，如果要移植到Java上来，非常浪费时间，而JNI可以支持Java程序与C语言编写的库进行交互，这样就不必要进行移植了。或者是与硬件、操作系统进行交互、提高程序的性能等，都可以使用JNI。需要注意的一点是需要保证本地代码能工作在任何Java虚拟机环境。

JNI的副作用

一旦使用JNI，Java程序将丢失了Java平台的两个优点：

- 1、程序不再跨平台，要想跨平台，必须不同的系统环境下程序编译配置本地语言部分。
- 2、程序不再是绝对安全的，本地代码的使用不当可能会导致整个程序崩溃。一个通用规则是，调用本地方法应该集中在少数的几个类当中，这样就降低了Java和其他语言之间的耦合。

JVM GC（垃圾回收机制）

详情请参考我的另外一篇文章：[Java性能优化之JVM GC（垃圾回收机制） - 知乎专栏](#)

JVM 常量池

JVM常量池也称之为运行时常量池，它是方法区（Method Area）的一部分。用于存放编译期间生成的各种字面量和符号引用。运行时常量池不要求一定只有在编译器产生的才能进入，运行期间也可以将新的常量放入池中，这种特性被开发人员利用比较多的就是String.intern()方法。

由“用于存放编译期间生成的各种字面量和符号引用”这句话可见，常量池中存储的是对象的引用而不是对象的本身。

常量池的好处

常量池是为了避免频繁的创建和销毁对象而影响系统性能，它也实现了对象的共享。

知乎

2、节省运行时间：比较字符串时，`==`比`equals()`快。对于两个引用变量，`==`判断引用是否相等，也就可以判断实际值是否相等。

双等号 (==) 的含义

基本数据类型之间使用双等号，比较的是数值。

复合数据类型（类）之间使用双等号，比较的是对象的引用地址是否相等。

八种基本类型的包装类和常量池

Byte、Short、Integer、Long、Character、Boolean、String这7种包装类都各自实现了自己的常量池。

```
//例子：
Integer i1 = 20;
Integer i2 = 20;
System.out.println(i1==i2);//输出TRUE
```

Byte、Short、Integer、Long、Character这5种包装类都默认创建了数值`[-128, 127]`的缓存数据。当对这5个类型的数据不在这个区间内的时候，将会去创建新的对象，并且不会将这些新的对象放入常量池中。

```
//IntegerCache.Low = -128
//IntegerCache.high = 127
public static Integer valueOf(int i) {
    if (i >= IntegerCache.low && i <= IntegerCache.high)
        return IntegerCache.cache[i + (-IntegerCache.low)];
    return new Integer(i);
}
```

```
//例子
Integer i1 = 200;
Integer i2 = 200;
System.out.println(i1==i2);//返回FALSE
```

Float 和Double 没有实现常量池。

String包装类与常量池

```
String str1 = "aaa";
```

当以上代码运行时，JVM会到字符串常量池查找 "aaa" 这个字面量对象是否存在？

存在：则返回该对象的引用给变量 **str1**。

不存在：则在堆中创建一个相应的对象，将创建的对象的引用存放到常量池中，同时将引用返回给




```
System.out.println(str1 == str2); //返回TRUE
```

因为变量`str1` 和`str2` 都指向同一个对象，所以返回`true`。

```
String str3 = new String("aaa");
System.out.println(str1 == str3); //返回FALSE
```

当我们使用了`new`来构造字符串对象的时候，不管字符串常量池中是否有相同内容的对象的引用，新的字符串对象都会创建。因为两个指向的是不同的对象，所以返回`FALSE`。

String.intern()方法

对于使用了`new` 创建的字符串对象，如果想要将这个对象引用到字符串常量池，可以使用`intern()`方法。

调用`intern()` 方法后，检查字符串常量池中是否有这个对象的引用，并做如下操作：

存在：直接返回对象引用给变量。

不存在：将这个对象引用加入到常量池，再返回对象引用给变量。

```
String interns = str3.intern();
System.out.println(interns == str1); //返回TRUE
```

假定常量池中都没有以上字面量的对象，以下创建了多少个对象呢？

```
String str4 = "abc"+"efg";
String str5 = "abcefg";
System.out.println(str4 == str5); //返回TRUE
```

答案是三个。第一个："abc"，第二个："efg"，第三个："abc"+"efg" ("abcefg")

`String str5 = "abcefg";` 这句代码并没有创建对象，它从常量池中找到了"abcefg" 的引用，所有`str4 == str5` 返回`TRUE`，因为它们都指向一个相同的对象。

什么情况下会将字符串对象引用自动加入字符串常量池？

//只有在这两种情况下会将对象引用自动加入到常量池

```
String str1 = "aaa";
String str2 = "aa"+"a";
```

//其他方式下都不会将对象引用自动加入到常量池，如下：

```
String str3 = new String("aaa");
String str4 = new StringBuilder("aa").append("a").toString();
StringBuilder sb = new StringBuilder();
sb.append("aa");
sb.append("a");
String str5 = sb.toString();
```

好了，JVM的基本原理就写到这里，以后更深入的了解后会再来补充。

参考文章：

[java中JVM的原理 - WitsMakeMen的专栏 - 博客频道 - CSDN.NET](#)

[JVM原理和优化 - ning109314的专栏 - 博客频道 - CSDN.NET](#)

[Java Jvm运行机制原理 - ZengTao - 博客频道 - CSDN.NET](#)

[Java常量池理解与总结](#)

[Java本地接口_百度百科](#)

编辑于 2017-03-21 01:17

Java 虚拟机 (JVM) Java

推荐阅读

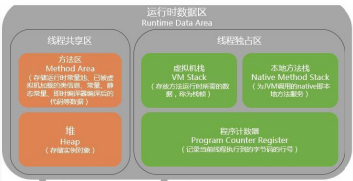


认识一下什么是JVM

一两月光 发表于java

JVM（生命周期、结构体系、堆、栈、运行机制&原理）核...

JVM的基础概念JVM的中文名称叫Java虚拟机，它是由软件技术模拟出计算机运行的一个虚拟的计算... JVM也充当着一个翻译官的角色，我们编写出的Java程序，是不能够被操作系统所直接识别的，... 君慕贤



Java虚拟机 (JVM) 你只要看这一篇就够了！

德鲁大叔 发表于JAVA后...

22 条评论

切换为时间排序

写下你的评论...



amazonove

2017-05-11

写的太好了

2



A1340399426

2019-11-02

String包装类与常量池




知乎

存在：则返回该对象的引用给变量 str1 。

不存在：则在堆中创建一个相应的对象，将创建的对象引用存放到常量池中，同时将引用返回给变量 str1 。

上边的这一段写的不对，没有在堆中创建对象，而是直接在常量池中创建的对象，字符串引用也保存在常量池中

👍 1

 For-Dream

2018-08-10

JRE应该还包含基本库.rt.jar，所以说JRE=JVM不是很准确把


👍 1

 图灵满分 回复 For-Dream

2019-03-22

rt的意识不是runtime的意思么，也就是JVM运行时需要用的包，是JVM的一部分哦

👍 赞

 换id

2018-03-06

想看解释执行这个过程的源码，从哪个文件开始看啊？openjdk的

👍 1

 图灵满分 回复 换id

2019-03-22

找到了吗？

👍 赞

 换id 回复 图灵满分

2019-03-25

模版解释器，上iteye搜R大找找

👍 赞

 对非当割人生几何

2018-03-05

字符串这块讲的不错，有点收获

👍 1

 言无心

2018-02-23

学习了

👍 1

 你头发乱了哦

2018-02-22

不错

👍 1

 吨一

2017-07-12

非常感谢

👍 1

 鹿笙

2020-05-15

我可以转发你的文章吗

👍 赞

 Bone

2019-08-19

其实安装jdk的时候可以选择不安装额外的jre



知乎

mark

👍 赞



Leslie Lee2018-09-29

“八种基本类型的包装类和常量池”下的代码例子中:

```
System.out.println(i1 = i2);
```

少了一个=号，应该是：

```
(i1 == i2)
```

👍 赞



Than best2018-08-12

问个小问题，既然当new一个字符串的时候，如果常量池中并没有相等的则添加到常量池，也就是说所有new的字符串常量池中都有与其相等的，那么intern有什么意义呢？（我是小白，比较菜）

👍 赞



易建军2021-09-08

a

👍 赞



易建军2021-09-08

a

👍 赞



手机摄影之美2021-03-10

写得非常棒

👍 赞



19002021-02-13

学到很多👏

👍 赞



mazhimazh2020-11-18

我在B站做了Java虚拟机HotSpot源代码分析的视频，哔哩哔哩(°- °)つ口 乾杯~ Bilibili，或者个人博客也有不少分析源代码文章，可以做为参考

👍 赞



哎哟哥哥 回复 mazhimazh02-09

哪里呀 学习下

👍 赞