Java 基础提升篇：Java 中 Native 关键字的作用

初遇

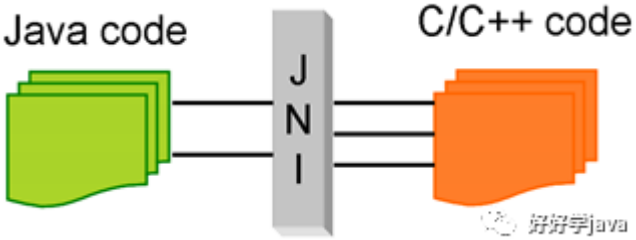
初次遇见 native 是在 java.lang.Object 源码中的一个 hashCode 方法：

```
public native int hashCode();
```

为什么有个 native 呢？这是我所要学习的地方。所以下面想要总结下 native。

认识 native 即 JNI,Java Native Interface

凡是一种语言，都希望是纯。比如解决某一个方案都喜欢就单单这个语言来写即可。Java 平台有个用户和本地 C 代码进行互操作的 API，称为 Java Native Interface (Java 本地接口)。



用 Java 调用 C 的“Hello，JNI”

我们需要按照下班方便的步骤进行：

1. **创建一个 Java 类**，里面包含着一个 native 的方法和加载库的方法 loadLibrary。HelloNative.java 代码如下：

```
public class HelloNative {
    static {
        System.loadLibrary("HelloNative");
    }
    public static native void sayHello();
    @SuppressWarnings("static-access")
    public static void main(String[] args) {
```

```
        new HelloNative().sayHello();
    }
}
```

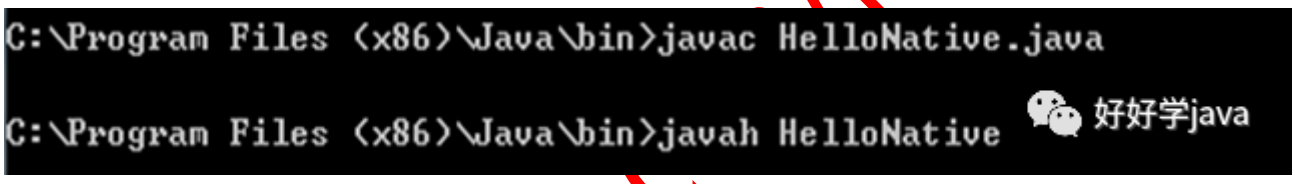
首先让大家注意的是 native 方法，那个加载库的到后面也起作用。native 关键字告诉编译器（其实是 JVM）调用的是该方法在外部定义，这里指的是 C。如果大家直接运行这个代码， JVM 会告之： “A Java Exception has occurred.” 控制台输出如下：

```
Exception in thread "main" java.lang.UnsatisfiedLinkError: no HelloNative in
java.library.path
   at java.lang.ClassLoader.loadLibrary(Unknown Source)
   at java.lang.Runtime.loadLibrary0(Unknown Source)
   at java.lang.System.loadLibrary(Unknown Source)
   at HelloNative.<clinit>(HelloNative.java:5)
```

这是程序使用它的时候，虚拟机说不知道如何找到 sayHello。

运行 javah，得到包含该方法的 C 声明头**文件.h**

将 HelloNative.java ，简单地 javac javah，如图



就得到了下面的 **HelloNative.h** 文件：

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class HelloNative */
#ifndef _Included_HelloNative
#define _Included_HelloNative
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     HelloNative
 * Method:    sayHello
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_HelloNative_sayHello
  (JNIEnv *, jclass);
#ifdef __cplusplus
}
```

```
#endif
#endif
```

jni.h 这个文件，在/%JAVA_HOME%include

2. 根据头文件，写 C 实现本地方法。

这里我们简单地实现这个 sayHello 方法如下：

```
#include "HelloNative.h"
#include <stdio.h>
JNIEXPORT void JNICALL Java_HelloNative_sayHello
{
    printf("Hello , JNI");
}
```

3. 生成 dll 共享库，然后 Java 程序 load 库，**调用**即可。

在 Windows 上，MinGW GCC 运行如下

```
gcc -m64  -Wl,--add-stdcall-alias -I"C:\Program Files\Java\jdk1.7.0_71\include" -
I"C:\Program Files\Java\jdk1.7.0_71\include\include\win32" -shared -o
HelloNative.dll HelloNative.c
```

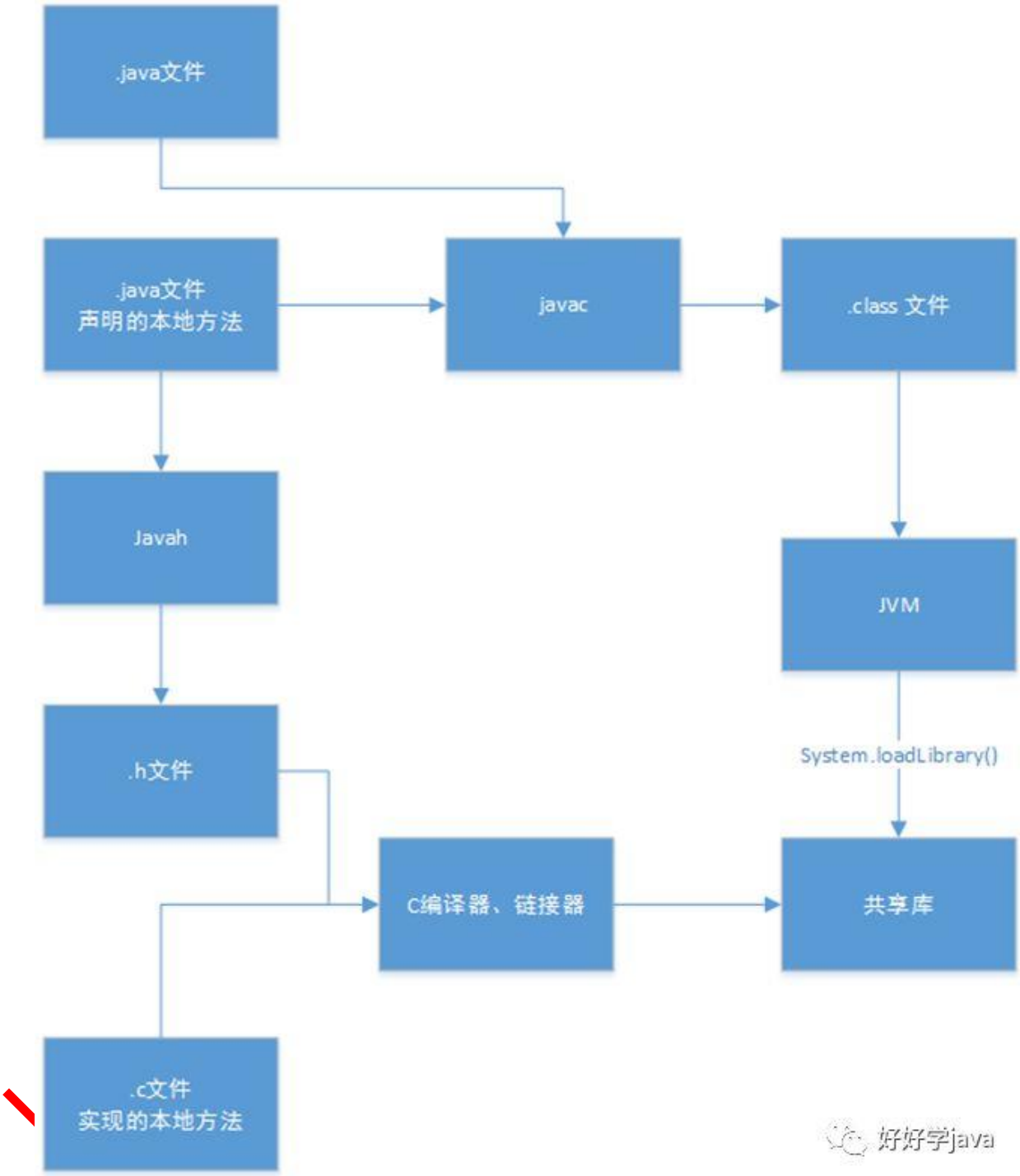
-m64 表示生成 dll 库是 64 位的。然后运行 **HelloNative**：

```
java HelloNative
```

终于成功地可以看到控制台打印如下：

```
Hello , JNI
```

JNI 调用 C 流程图



其他介绍

native 是与 C++联合开发的时候用的！java 自己开发不用的！

使用 native 关键字说明这个方法是原生函数 ,也就是这个方法是用 C/C++语言实现的，并且被编译成了 DLL，由 java 去调用。

这些函数的实现体在 DLL 中，JDK 的源代码中并不包含，你应该是看不到的。对于不同的平台它们也是不同的。这也是 java 的底层机制，实际上 java 就是在不同的平台上调用不同的 native 方法实现对操作系统的访问的。

- native 是用做 java 和其他语言（如 c++）进行协作时用的，也就是 native 后的函数的实现不是用 java 写的
- 既然都不是 java，那就别管它的源代码了。

native 的意思就是通知操作系统，这个函数你必须给我实现，因为我要使用。所以 native 关键字的函数都是操作系统实现的，java 只能调用。

java 是跨平台的语言，既然是跨了平台，所付出的代价就是牺牲一些对底层的控制，而 java 要实现对底层的控制，就需要一些其他语言的帮助，这个就是 native 的作用了。

Java 不是完美的，Java 的不足除了体现在运行速度上要比传统的 C++慢许多之外，Java 无法直接访问到操作系统底层（如系统硬件等），为此 Java 使用 native 方法来扩展 Java 程序的功能。

可以将 native 方法比作 Java 程序同 C 程序的接口，其实现步骤：

1) 在 Java 中声明 native()方法，然后编译；
2) 用 javah 产生一个.h 文件；
3) 写一个.cpp 文件实现 native 导出方法，其中需要包含第二步产生的.h 文件（注意其中又包含了 JDK 带的 jni.h 文件）；
4) 将第三步的.cpp 文件编译成动态链接库文件；
5) 在 Java 中用 System.loadLibrary()方法加载第四步产生的动态链接库文件，这个 native()方法就可以在 Java 中被访问了。

JAVA 本地方法适用的情况

1) 为了使用底层的主机平台的某个特性，而这个特性不能通过 JAVA API 访问
2) 为了访问一个老的系统或者使用一个已有的库，而这个系统或这个库不是用 JAVA 编写的
3) 为了加快程序的性能，而将一段时间敏感的代码作为本地方法实现。