Android高级开发正式课

码牛学院-用代码码出牛逼人生

android人员的专属JVM讲解

01-JVM虚拟机运行时数据区

技术点：

1.Hotspot、Dalvik、ART关系与对比

2.JVM的跨语言与字节码

3.运行时数据区中堆栈的职责

4.栈区存储结构与运行原理

5.栈帧内部结构解析

6.Jclasslib与HSDB工具应用分析

码牛学院Android讲师介绍

码牛学院-Kerwin老师
系统架构师、技术总监

◆10年互联网行业从业经验，架构师
精通JAVA,C,C++,Android,IOS

前华为工程师，后出任两家公司技术总监，高校外聘讲师，省公安厅电子物证鉴定专家

拥有多个大型分布式系统架构设计与实施和移动终端系统架构设计经验

有丰富的分布式，高并发实战经验，
开发过多套企业级自定义框架
擅长系统底层架构，移动终端系统架构

课程安排

01

JVM是一种规范

02

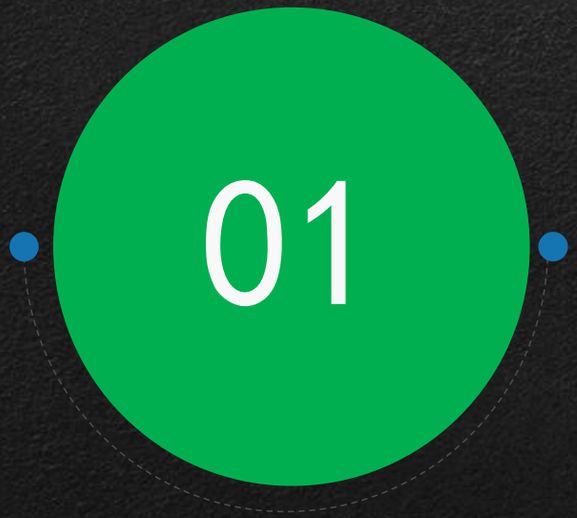
方法调用过程

03

对象分配过程

04

运行时数据区下管理带来的内存问题

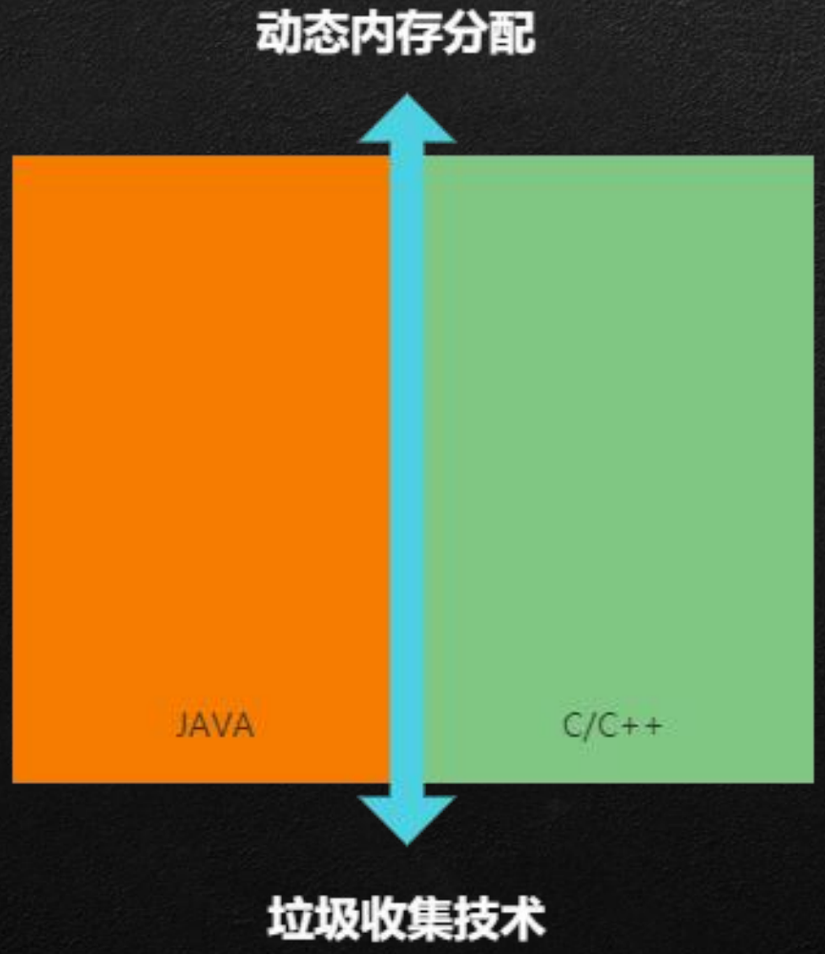


到底什么是JVM？为什么JVM是一种规范？

下面的场景你经历过哪些？

1. 程序有莫名的卡顿找不到原因？
2. 程序运行过程中突然出现OOM现象！
3. 每次面试之前先找一堆资料背了JVM相关问题但是，面试过程中问的问题与背的内容存在偏差
4. 写出来的代码质量也并不高！

JAVA对比与C所带来的问题



JVM面向人群

高工、架构、顾问、CTO等一切工作岗位不是写业务代码的都需要掌握的硬性要求

JVM是一种规范

1. JVM到底是什么？
2. JAVA所谓的跨语言性是什么？
3. 为什么说JVM是一种规范？

Java程序的执行过程

一个 Java 程序，首先经过 javac 编译成 .class 文件，然后 JVM 将其加载到方法区，执行引擎将会执行这些字节码。执行时，会翻译成操作系统相关的函数。JVM 作为 .class 文件的翻译存在，输入字节码，调用操作系统函数。

过程如下：Java 文件->编译器>字节码->JVM->机器码。

JVM 全称 Java Virtual Machine，也就是我们耳熟能详的 Java 虚拟机。它能识别 .class后缀的文件，并且能够解析它的指令，最终调用操作系统上的函数，完成我们想要的操作

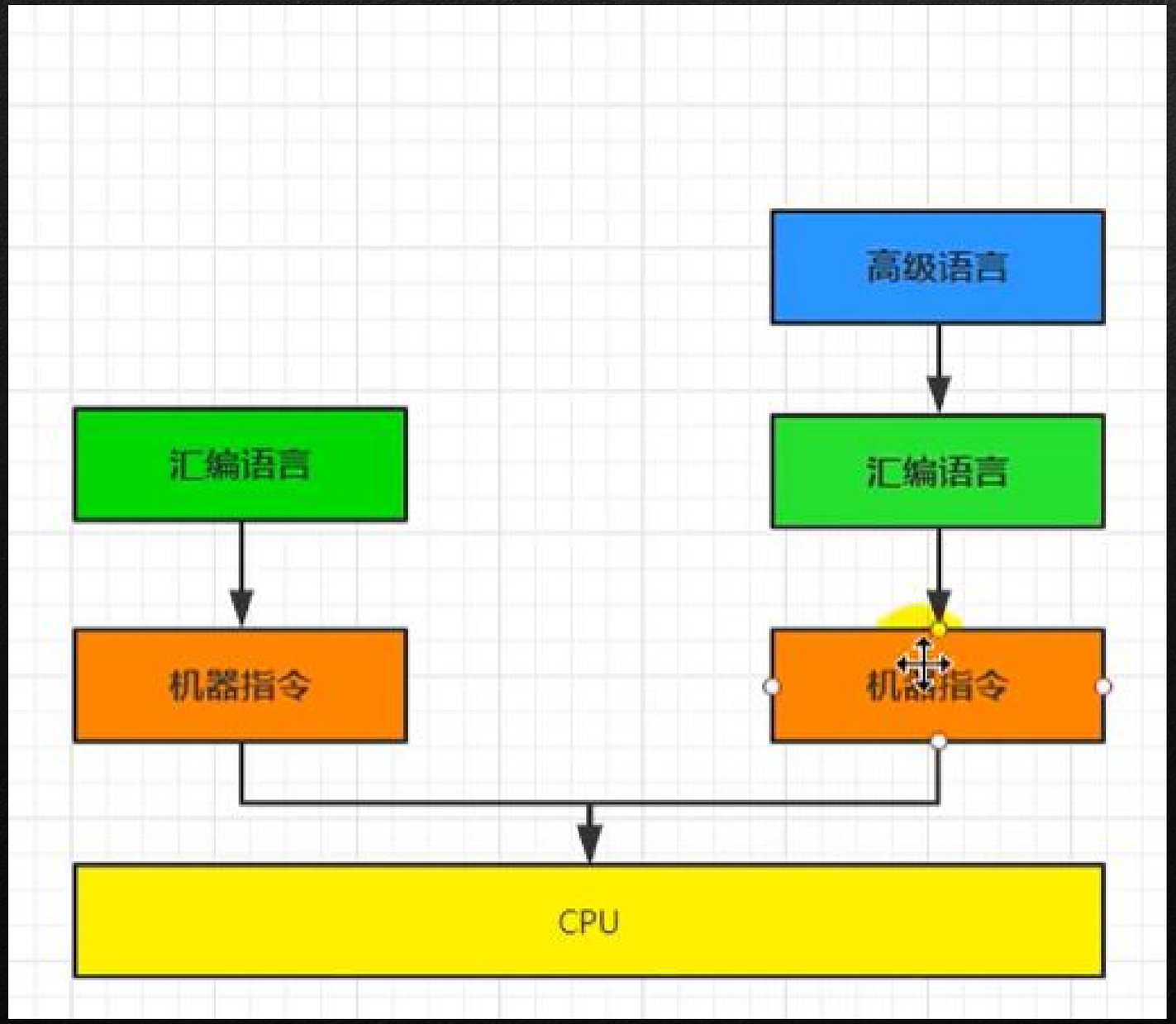
字节码文件与JVM

我们平时说的JAVA字节码，指的是JAVA语言编译（通过javac编译.java后缀文件）成的字节码，准确的说任何能在JVM平台上执行的字节码格式都是一样的，所以应该统称为JVM字节码

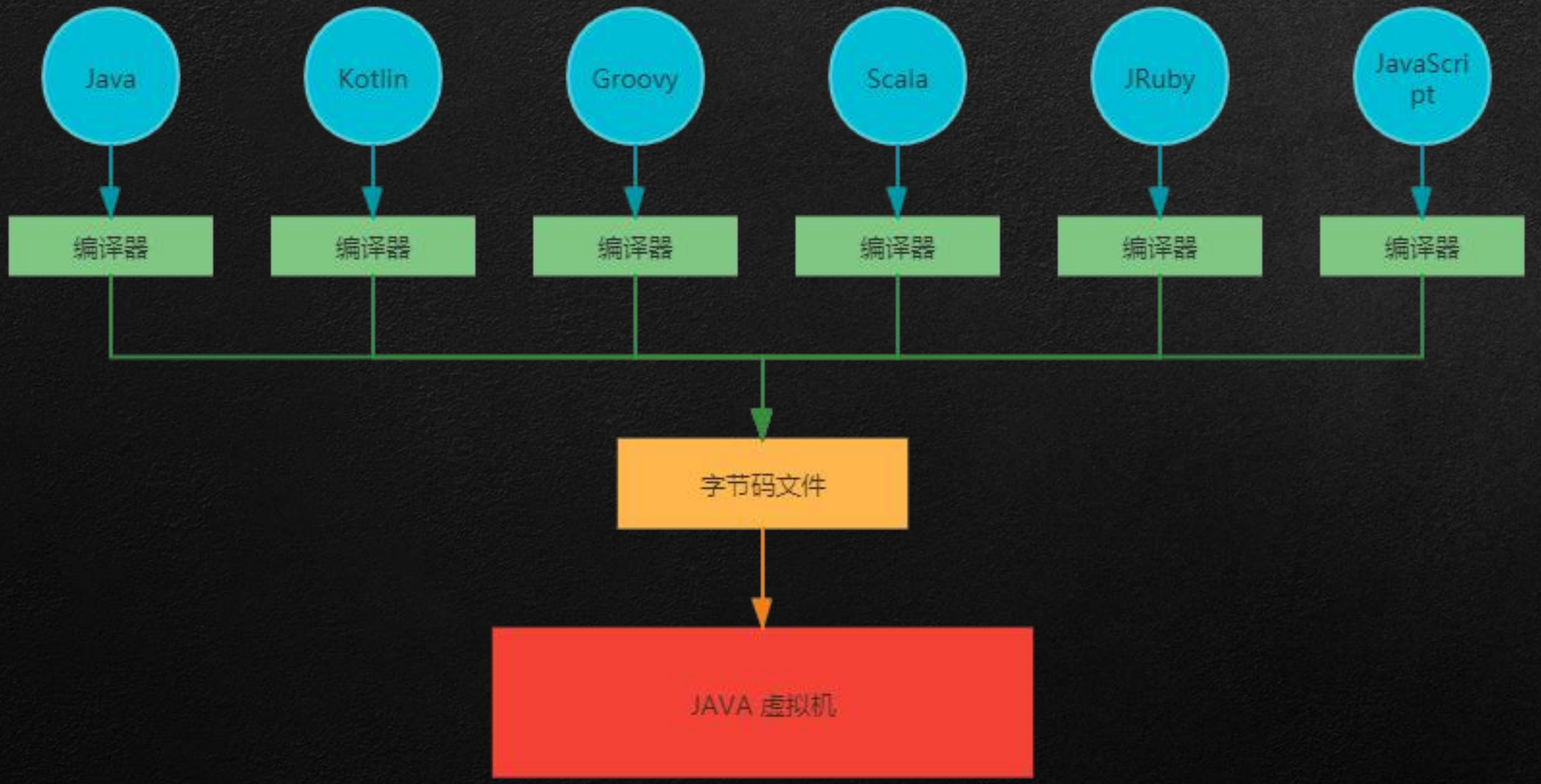
不同的编译器，可以编译出相同的字节码文件，字节码文件也可以在不同的JVM上运行

JAVA虚拟机与JAVA语言并没有直接联系，他只是特定的二进制文件格式.class文件有所关联，CLASS文件中包含JVM虚拟机指令集（bytecodes）和符号表，还有一些其他辅助信息。

高级语言与汇编语言&机器指令



JAVA跨语言性的设计思路



栈指令集架构与寄存器指令集架构

JAVA编译器指令流是基于栈的指令集架构，而另一种指令集架构为基于寄存器的指令集架构

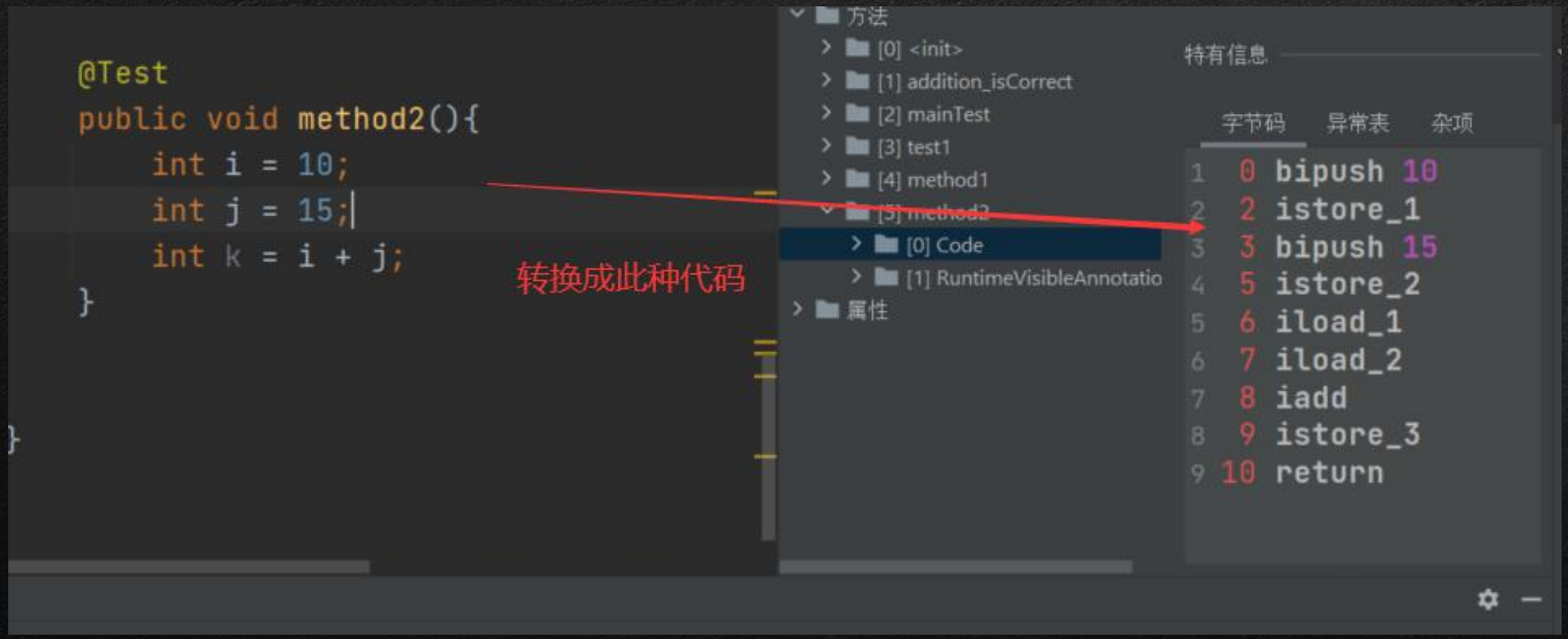
基于栈的指令集架构特点：

- **设计与实现简单，适用于资源受限系统**
- **避开寄存器的分配问题：使用0地址指令方式**
- **指令流中的指令操作过程基于栈，且位数小（8位），编译器容易实现**
- **不需要硬件支持，可移植性好**

基于寄存器的指令集架构特点：

- x86二进制指令集（区别于栈的8位，此处是16位）：android中Davlik使用的是这种架构
- 依赖于硬件，可移植性插
- 性能优秀和执行更加高效
- 花费更少时间去执行一个操作
- 基于寄存器架构的指令往往都以1~3地址指令为主，而基于栈则省却地址指令操作，都基于栈区完成

栈指令集



寄存器指令集
mov eax,10
add eax,15

Hotspot虚拟机

HotSpot:

隶属：sun

HotSpot历史发展版本：

1.最初由Longview Technologies设计开发
2.97年被Sun公司收购，09年Oracle收购sun
3.JDK1.3开发Hotspot成为默认虚拟机
4.现阶段占据JAVA语言虚拟机市场的绝对地位
5.一般面试所有提到的JVM虚拟机都默认指代的是Hotspot虚拟机

Dalvik虚拟机&ART虚拟机

Dalivk VM:

隶属：Google

发展历史：

应用于Android系统，并且在Android2.2中提供了JIT，发展迅猛

Dalvik是一款不是JVM的JVM虚拟机。本质上他没有遵循与JVM规范

不能直接运行java Class文件

他的结构基于寄存器结构，而不是JVM栈架构

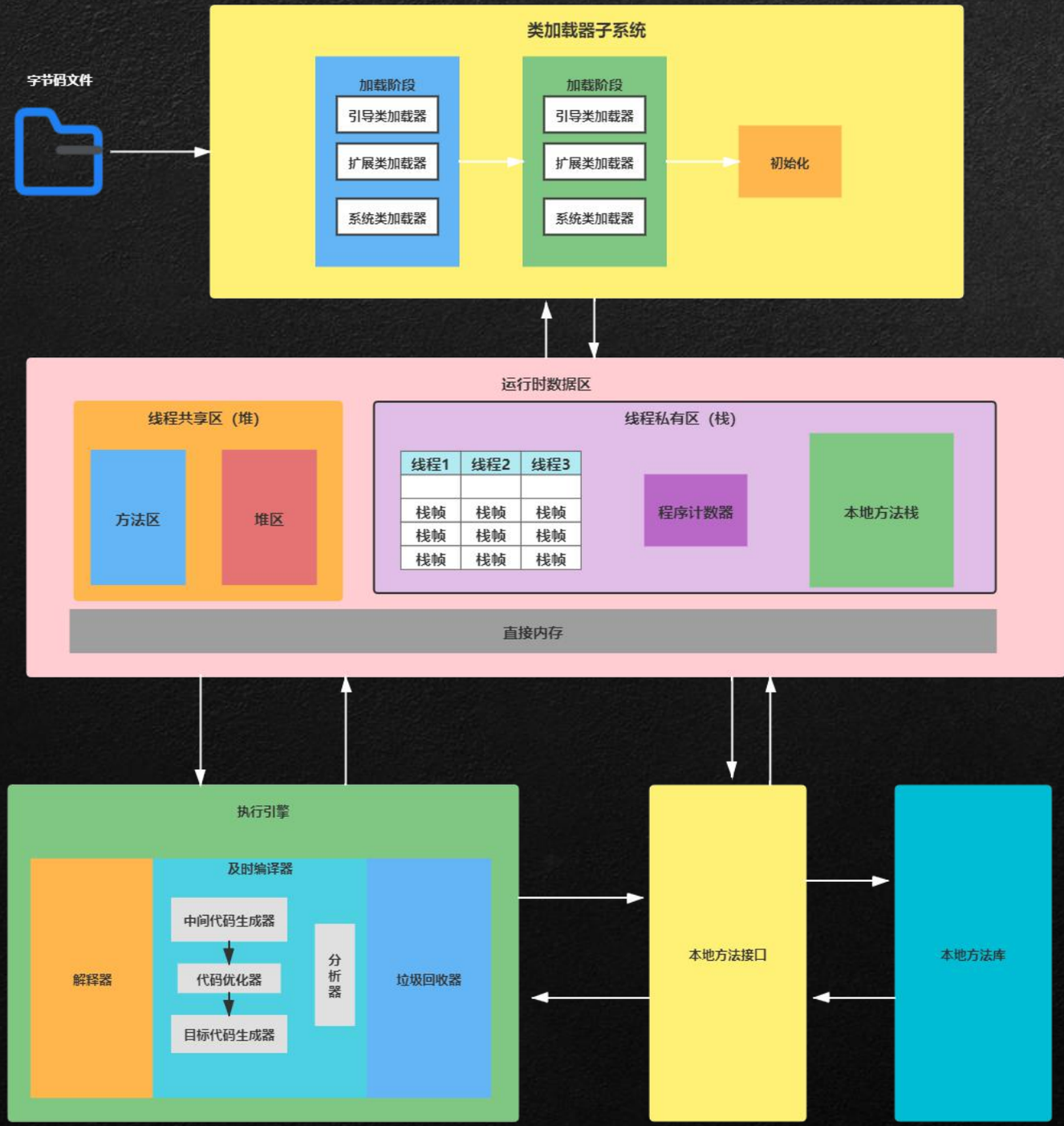
执行的是编译后的Dex文件，执行效率较高

与Android5.0后被ART替换

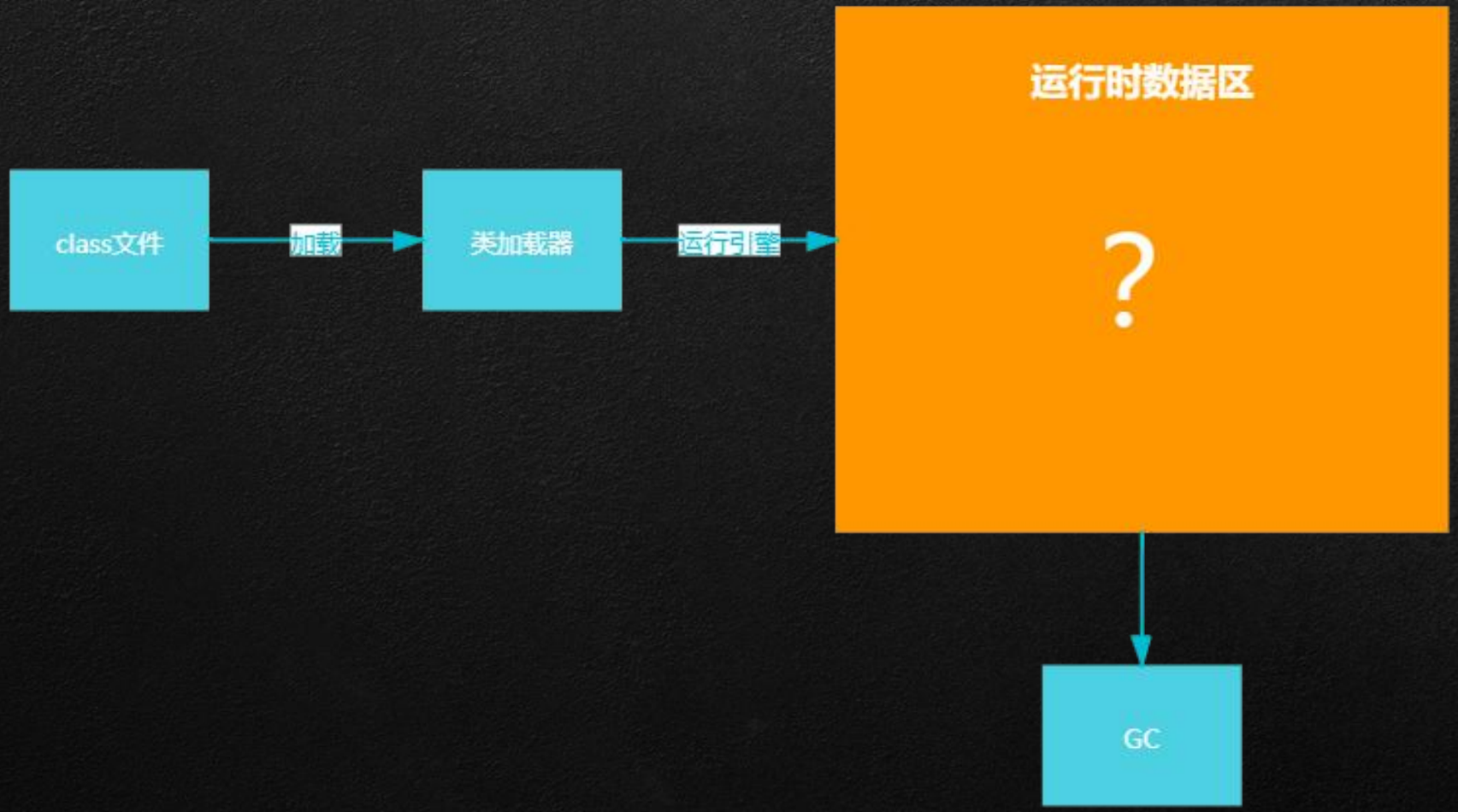
JVM组成部分及架构示意

JVM三大构成组件

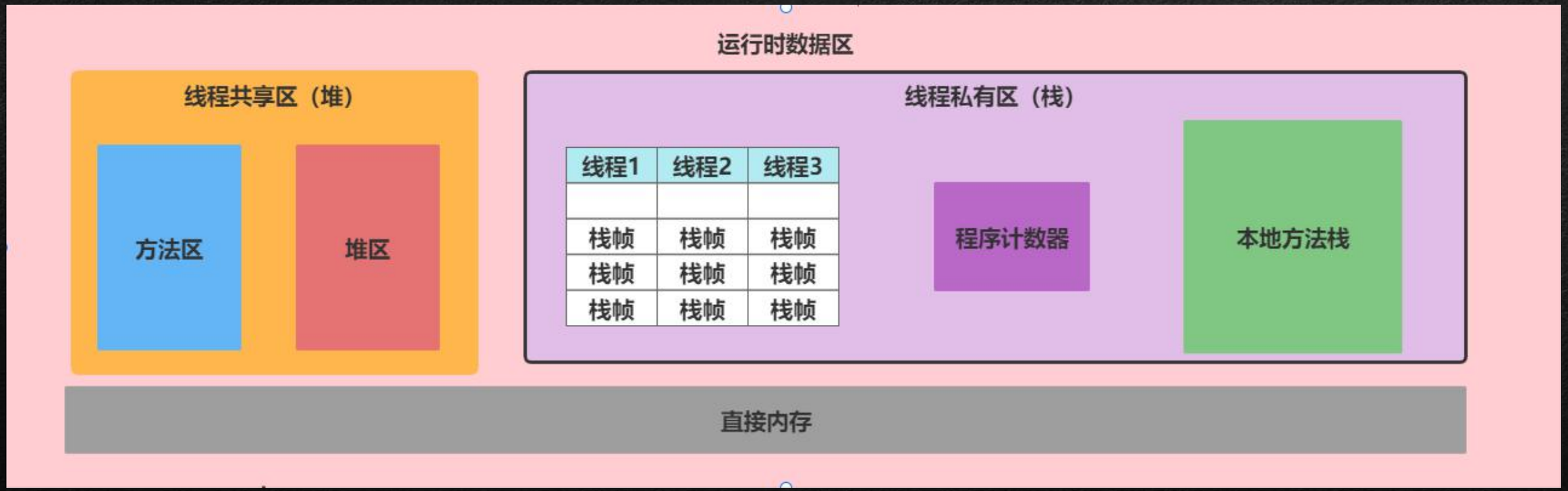
- 类加载器：将编译好的class文件加载到JVM进程中
- 运行时数据区：存放系统执行过程中产生的数据
- 执行引擎：用来执行汇编及当前进程内所要完成的一些具体内容


类加载器子系统
字节码文件
加载阶段
引导类加载器
扩展类加载器
系统类加载器
加载阶段
引导类加载器
扩展类加载器
系统类加载器
初始化
运行时数据区
线程共享区（堆）
方法区
堆区
线程私有区（栈）
线程1
线程2
线程3
栈帧
栈帧
栈帧
栈帧
栈帧
栈帧
栈帧
栈帧
栈帧
程序计数器
本地方法栈
直接内存
执行引擎
解释器
及时编译器
中间代码生成器
代码优化器
目标代码生成器
分析器
垃圾回收器
本地方法接口
本地方法库

引入运行时数据区概念



运行时数据区结构


运行时数据区
线程共享区（堆）
方法区
堆区
线程私有区（栈）
线程1
线程2
线程3
栈帧
栈帧
栈帧
栈帧
栈帧
栈帧
栈帧
栈帧
栈帧
程序计数器
本地方法栈
直接内存

堆栈在内存中的职责

栈是运行时的处理单位，而堆是运行时的存储单位!
即：

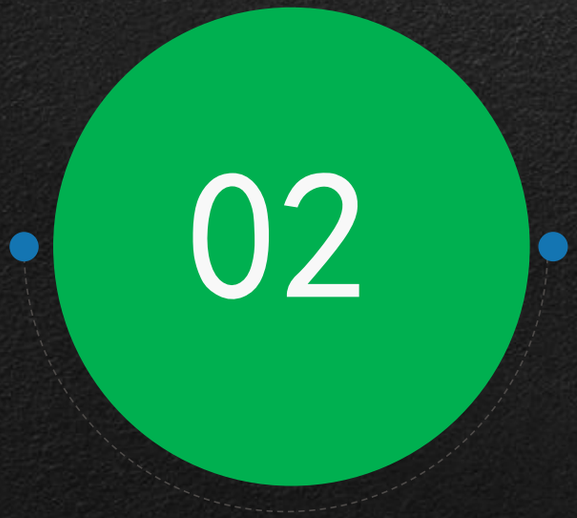
栈是用来解决程序运行问题，如程序如何运行，如何去处理数据，方法是怎么执行的

堆是用来解决数据存储问题，数据放哪、怎么放

荷包蛋辣椒炒肉的用料

肥肉	25g	瘦肉	100g
青辣椒	150g	红辣椒	50g
鸡蛋	2个	姜片	2片
大蒜	4瓣	葱段	1根
豆豉	少量	料酒	15ml
老抽	15ml	生抽	15ml
蚝油	5ml	盐	适量

堆栈在内存中的职责



运行时数据区解析，方法调用全过程

虚拟机栈基本信息

虚拟机栈是什么？

承载方法调用的过程中产生的数据容器，随线程开辟，为线程私有

作用：

他主管java方法运行过程中所产生的值变量、运算结果、方法的调用与返回等信息管理

主核心：

局部变量、计算结果

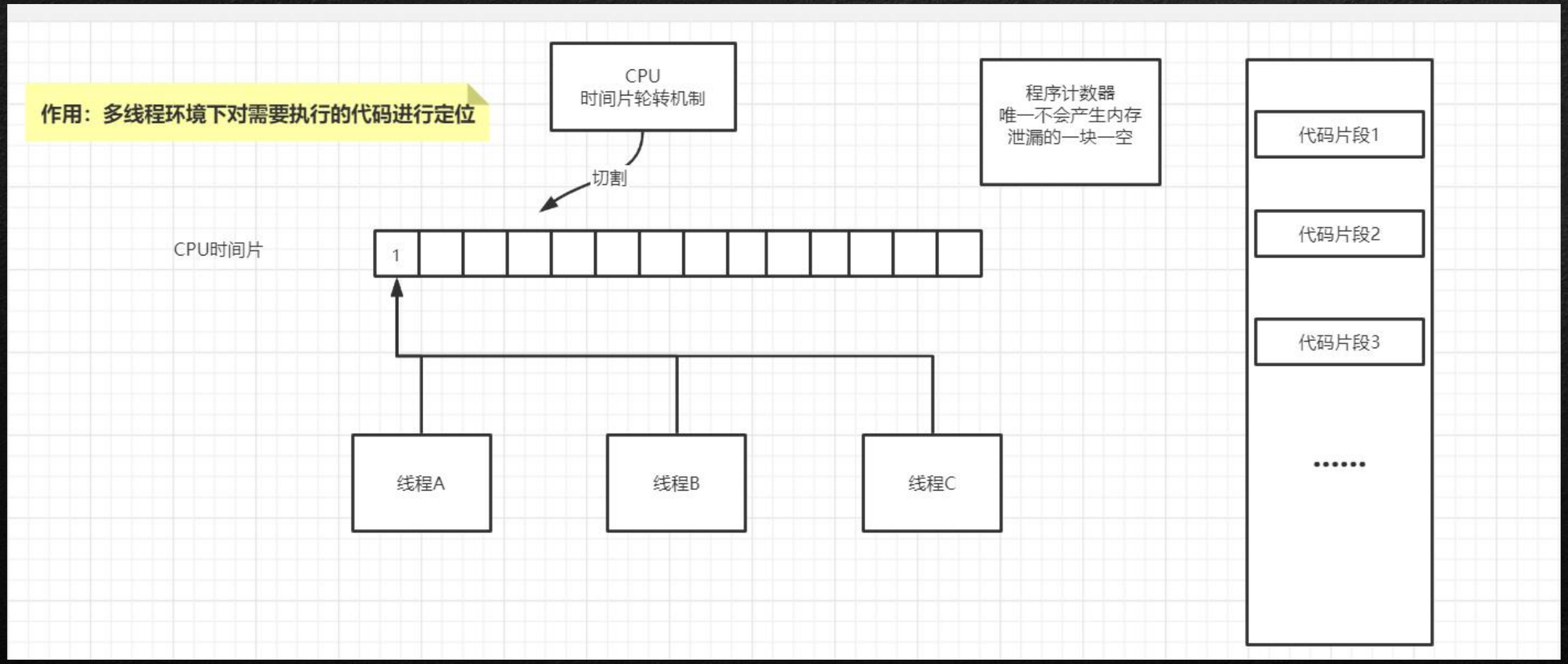
结构作用：

栈结构的应用能产生一种快速有效的分配方案，访问速度仅次于程序计数器

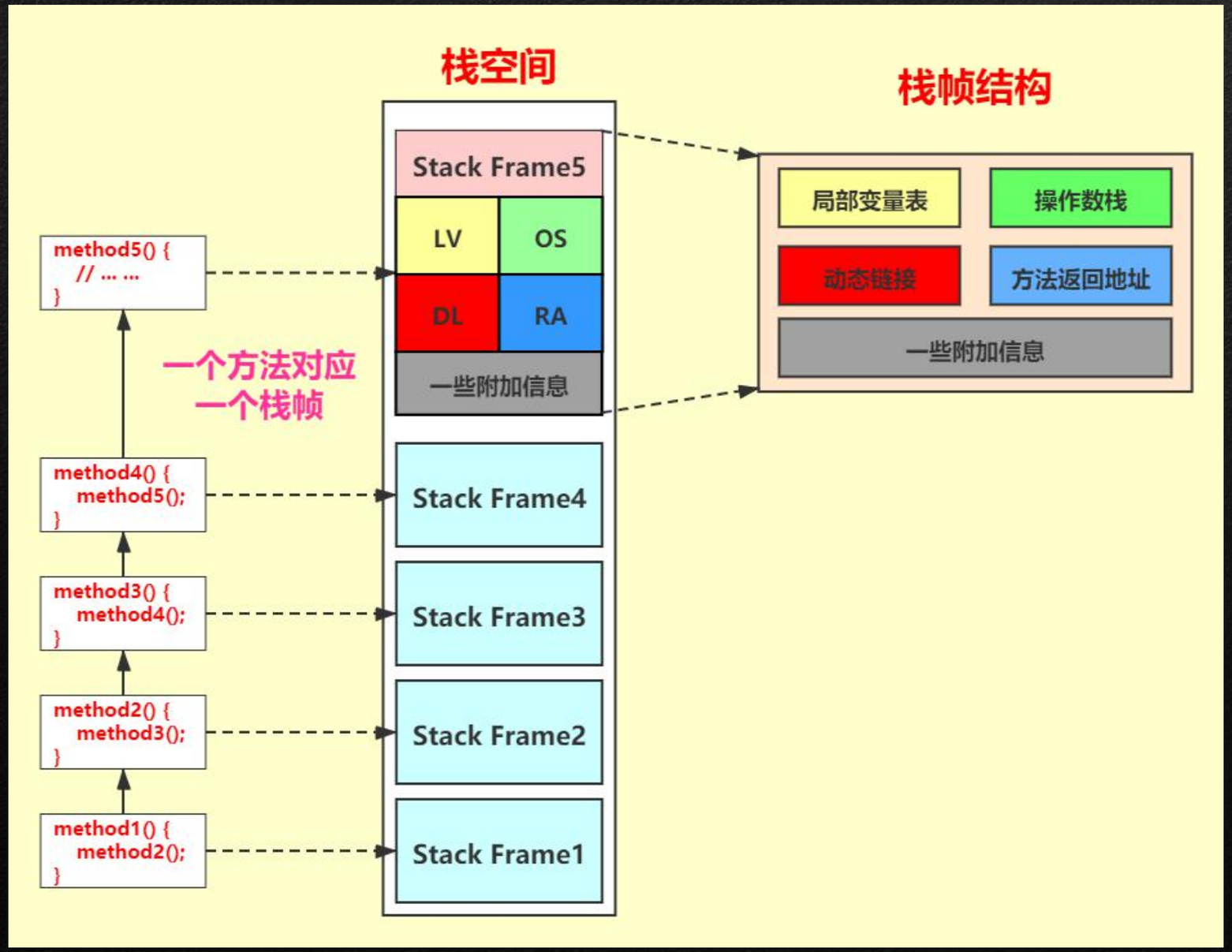
JAVA直接堆栈操作只有两个：出栈、入栈

此种应用不需要有GC设定

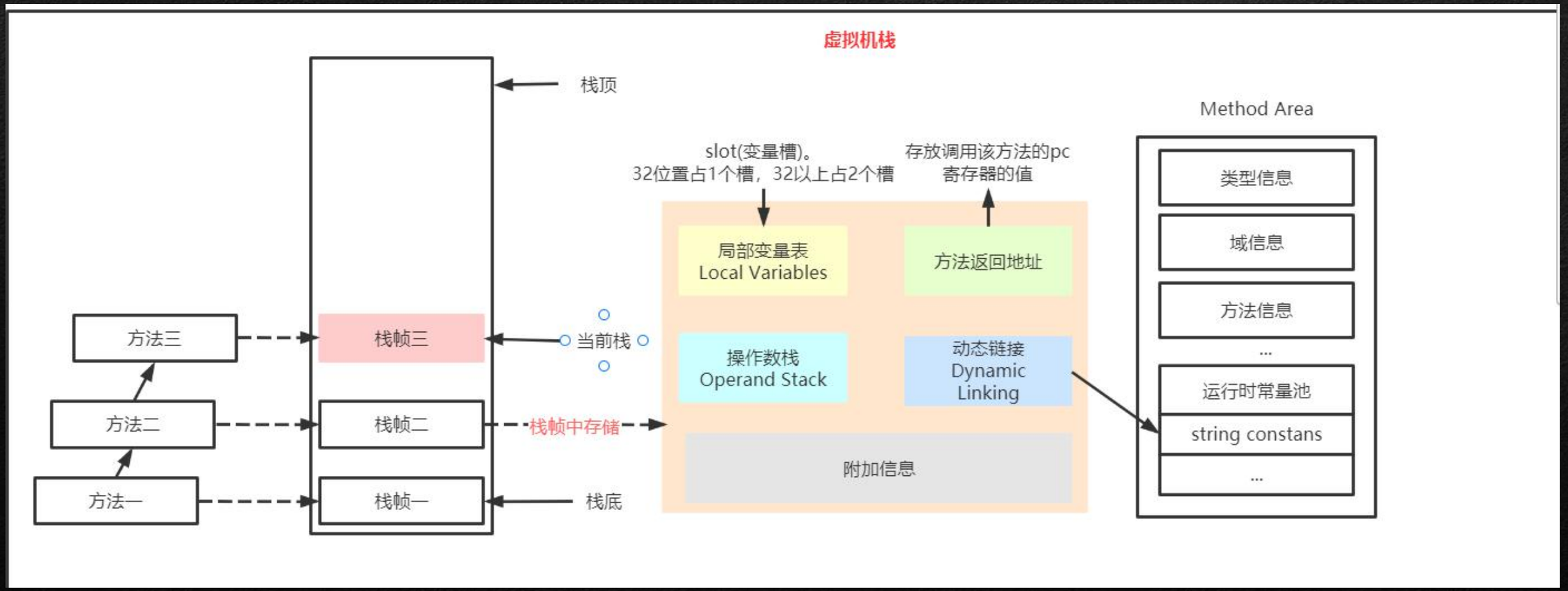
程序计数器/PC寄存器


作用：多线程环境下对需要执行的代码进行定位
CPU
时间片轮转机制
切割
CPU时间片
1
线程A
线程B
线程C
程序计数器
唯一不会产生内存
泄漏的一块一空
代码片段1
代码片段2
代码片段3
.....

栈区存储结构与运行原理



栈帧内部结构解析


虚拟机栈
栈顶
slot(变量槽)。
32位置占1个槽，32以上占2个槽
存放调用该方法的pc
寄存器的值
局部变量表
Local Variables
方法返回地址
操作数栈
Operand Stack
动态链接
Dynamic
Linking
附加信息
方法三
方法二
方法一
栈帧三
栈帧二
栈帧一
当前栈
栈帧中存储
栈底
Method Area
类型信息
域信息
方法信息
...
运行时常量池
string constans
...

局部变量表

局部变量表也被称之为局部变量数组或者本地变量表

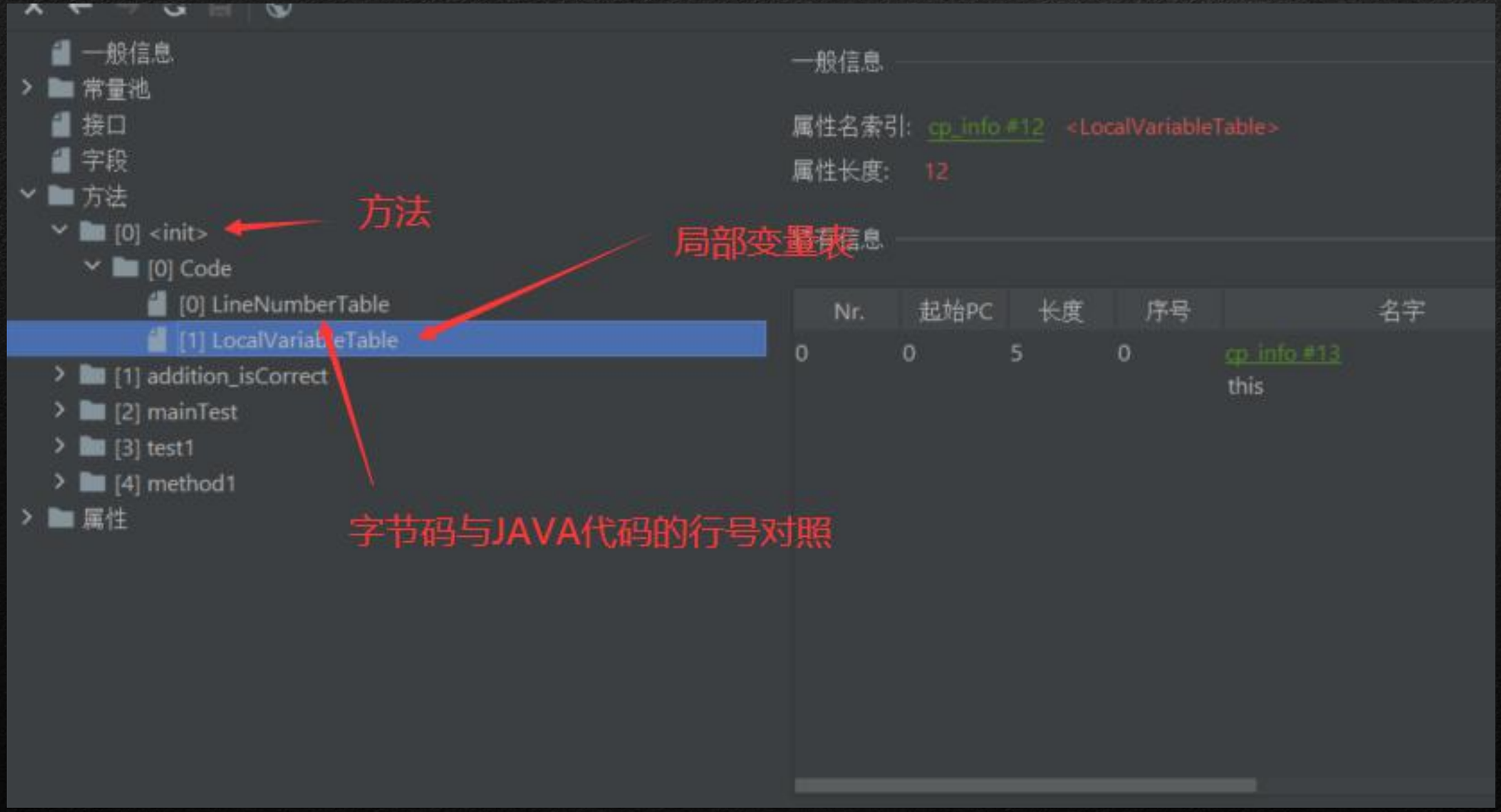
定义为一个数字数组，主要用于存储方法参数和定义方法体内的局部变量

由于局部变量表是建立在线程的栈上，是线程的私有数据，因此不存在数据安全问题

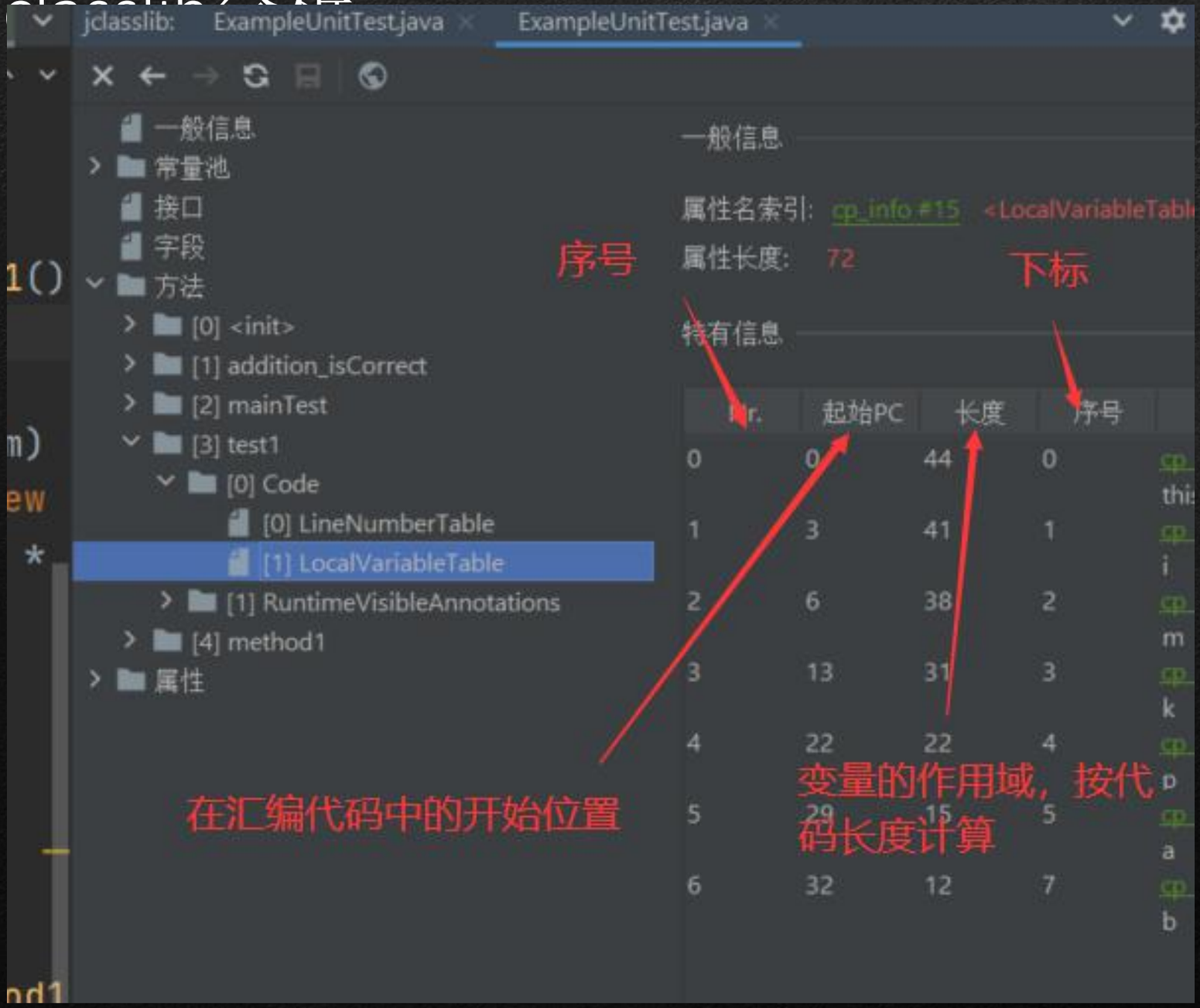
局部变量表需要的容量大小是在编译器确定下来的，病保存在方法的code属性的maximum local variables数据项中，运行期间局部变量表大小不变

方法嵌套调用的次数由栈的大小决定，局部变量表决定着栈帧的大小，这里是在编译期就会确定下来

局部变量表-jclasslib分析



局部变量表-jclasslib分析



关于局部变量表slot（变量槽）

slot是局部变量表的基础单位

在表中，32位类型数据占用一个slot，　64位数据占用2位

slot重复利用问题

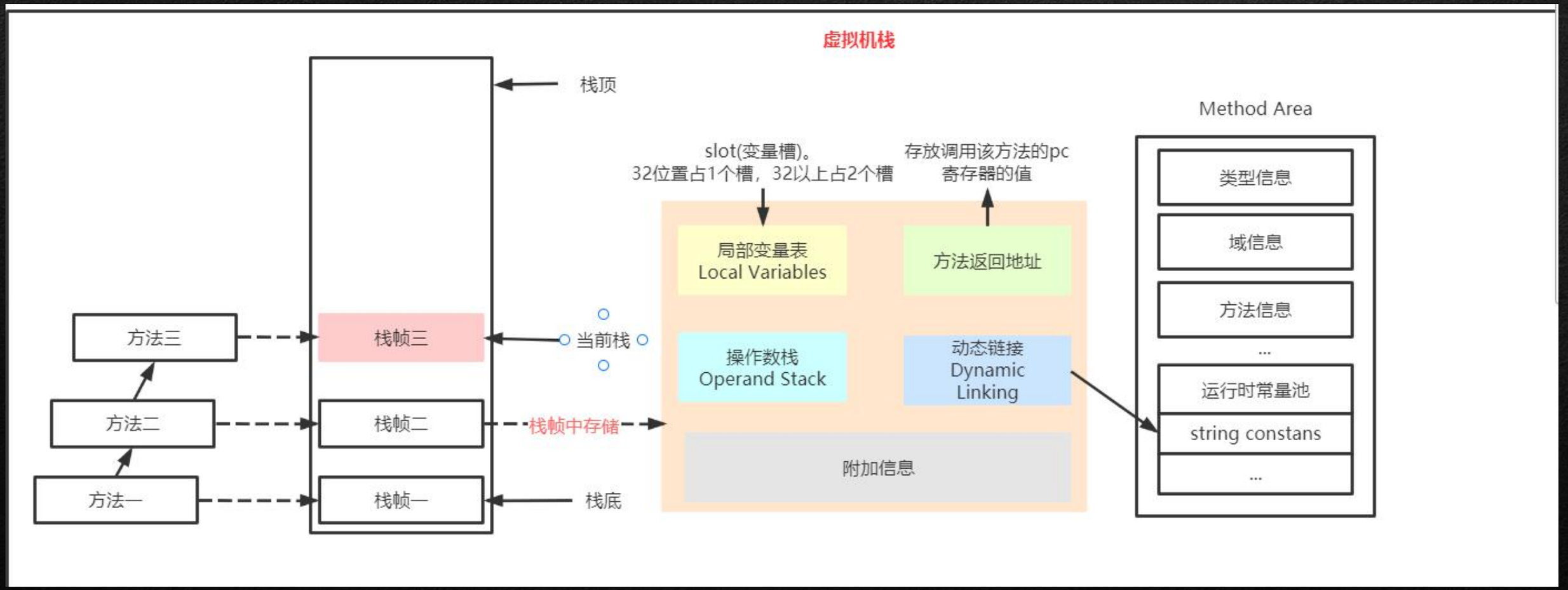
局部变量表中slot是可以重用的，如果一个局部变量过了其他作用域，那么其作用域之后声明的新的局部变量有可能会复用这个slot，以便于节省资源

操作数栈

每一个独立的栈帧中除了包含局部变量表之外还包含一个后劲先出的操作数栈，

作用：在方法执行过程中根据字节码指令，往栈中写入数据或者提取数据
某些字节码指令将值压入操作数栈，其余的字节码指令将操作数取出栈，使用他们后再把结果压入其中
比如：复制、交换、求和、求余等操作

操作数栈-代码追踪分析

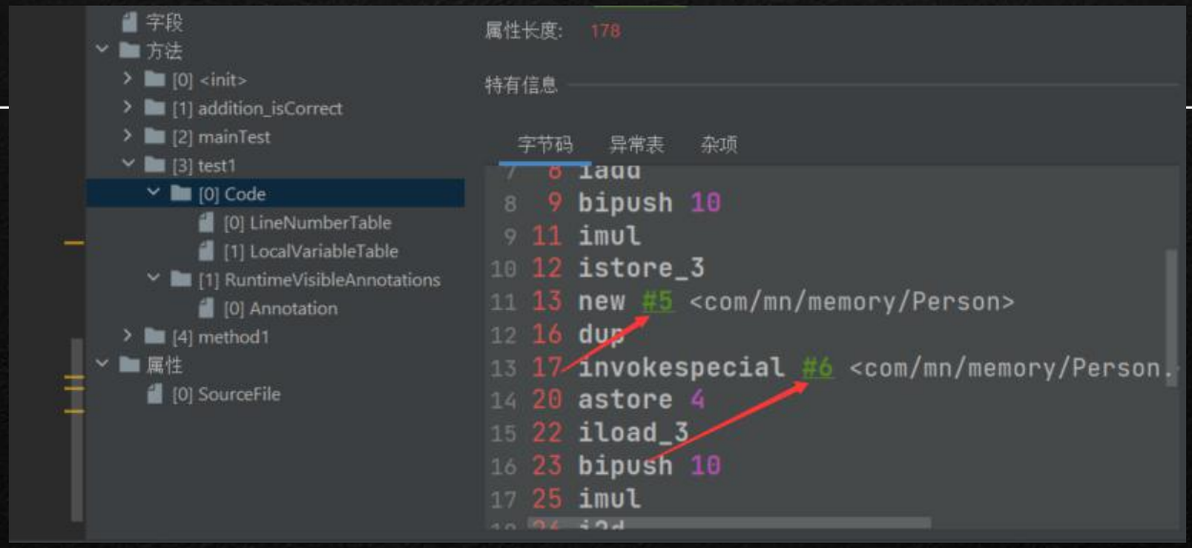

虚拟机栈
栈顶
slot(变量槽)。
32位置占1个槽，32以上占2个槽
存放调用该方法的pc
寄存器的值
局部变量表
Local Variables
方法返回地址
操作数栈
Operand Stack
动态链接
Dynamic
Linking
附加信息
方法三
方法二
方法一
栈帧三
栈帧二
栈帧一
当前栈
栈帧中存储
栈底
Method Area
类型信息
域信息
方法信息
...
运行时常量池
string constans
...

动态链接

每一个栈帧内部都包含一个执行运行时常量池中该栈帧所述方法的引用。包含这个引用的目的是为了支持当前方法的代码能够实现动态链接（invokeDynamic指令）

在Java源文件被编译到字节码文件中是，所有的变量和方法引用都作为符号引用保存在class文件的常量池里。

例：描述一个方法调用另外一个方法是，就是通过常量池中的执行方法符号引用来标识，那么动态链接的作用就是为了将这些符号引用转换为调用方法的直接引用



方法返回地址

存放调用方法的PC寄存器的值

一个方法的结束，有两种方式：

- 正常执行完成
- 出现未处理的异常，飞正常退出

无论通过那种方式退出，在方法退出后返回到该方法被调用的位置。方法正常退出是，调用者的PC寄存器的值作为返回地址，即调用该方法的指令的吓一条指令的地址。

异常表

而通过异常退出的，返回地址是要通过异常表来确定，栈帧中一般不会保存着部分信息，通过异常完成的出口退出的不会给他的上层调用者产生任何的返回值
只要在本方法中没有搜索到匹配的异常处理器就会异常退出

异常表如下所示：

```
public void method1();
  descriptor: ()V
  flags: ACC_PUBLIC
  Code:
    stack=1, locals=2, args_size=1
       0: aload_0
       1: invokespecial #2                  // Method method2:()V
       4: goto          12
       7: astore_1
       8: aload_1
       9: invokevirtual #4                  // Method java/io/IOException.printStackTrace:()V
      12: return
    Exception table:
       from    to  target type
           0     4     7   Class java/io/IOException
```

HSDB工具应用

脚本编写：
cd C:\Program Files\Java\jdk1.8.0_202\lib
java -cp .\sa-jdi.jar sun.jvm.hotspot.HSDB

坑点：UnsatisfiedLinkError异常

原因:在JDK目录中缺失sawindbg.dll文件
解决方案：
把自己Java\jre\bin目录下sawindbg.dll 粘贴到Java\jdk1.8.0_111\jre\bin 下既ok。

线程共享区域

方法区/永久代/元空间
- 类信息
- 常量
- 静态变量
- 及时编译器编译后的代码

堆区
- 对象示例
- 数组

方法区

《深入理解java虚拟机》一书中对于Method Area存储内容描述如下所示：、
他用于存储已被虚拟机加载的类型信息、常量、静态变量、及时编译器编译后的代码缓存等

类型信息：

包含类class\接口interface\枚举enum\注解annotation，JVM必须在方法区总存储以下类型信息

1. 这个类型的完整有效名称
2. 这个类型直接父类的完整邮箱名称
3. 这个类型的修饰符（public,abstract.final的某个子集）
4. 这个类型的直接接口的一个有序列表

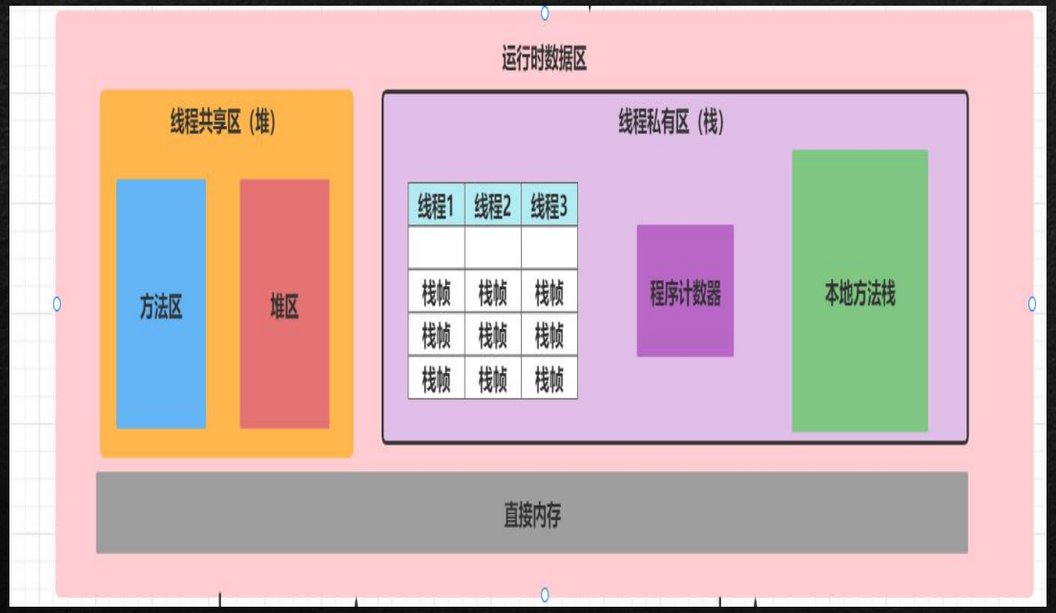
域（Field）信息：

JVM必须在方法区中保存所有与的相关信息

方法信息

1. 执行字节码
2. 本地变量表
3. 操作数栈
4. 动态链接
5. 方法出口
6. 异常表

直接内存



直接内存不是虚拟机运行时数据区的一部分，也不是JAVA虚拟机规范中定义的内存区域

这块区域会被频繁使用，在java堆内dictiByteBuffer对象直接引用操作

这块内存不收java堆的大小限制，但是受本机总内存的限制，可以通过maxDirectMemorySize来设置

深入底层探寻运行时数据区

深入辨析堆栈

功能

以栈帧方式存储方法的调用过程，病存储方法调用过程中产生的数据以及对象的引用变量，其内存分配在栈上，变量出作用域自动销毁，因为栈没有GC概念

堆内存用来存储java中的对象，无论是成员变量、局部变量、还是类变量，他们指向的对象都存储在堆内存中

线程独享与线程共享

栈内存属于单个线程，每个线程都会有一个栈内存，其存储变量只能在其所述线程中课件，即栈内存可以理解成线程私有内存

堆内存对所有对象进行共享课件，可以被所有线程访问

空间大小

栈内存要远远小于堆内存（默认1M），栈的深度是有限制的，可能发生StackOverFlowError问题

内存溢出问题汇总

内存溢出概念：

无法分配内存，可用内存不足，无法完成内存分配

内存溢出场景

栈溢出

堆溢出

方法区溢出

本机直接内存溢出

THANK YOU

码牛学院-用代码码出牛逼人生

谢谢观看