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

05-字节码文件与类加载

技术点：

1.解释执行&JIT&AOT

2.Class文件结构与Dex文件结构

3.“类”的生命周期

4.类加载器详解

5.热修复原理

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

Dalvik虚拟机与ART虚拟机

02

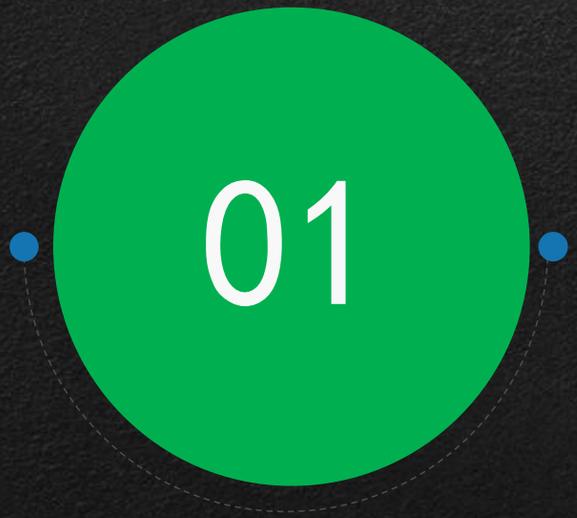
Art虚拟机运行时数据区

03

内存抖动与内存泄漏

04

案例演示



Dalvik虚拟机与ART虚拟机

Dalvik虚拟机&ART虚拟机与Hotspot区别

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

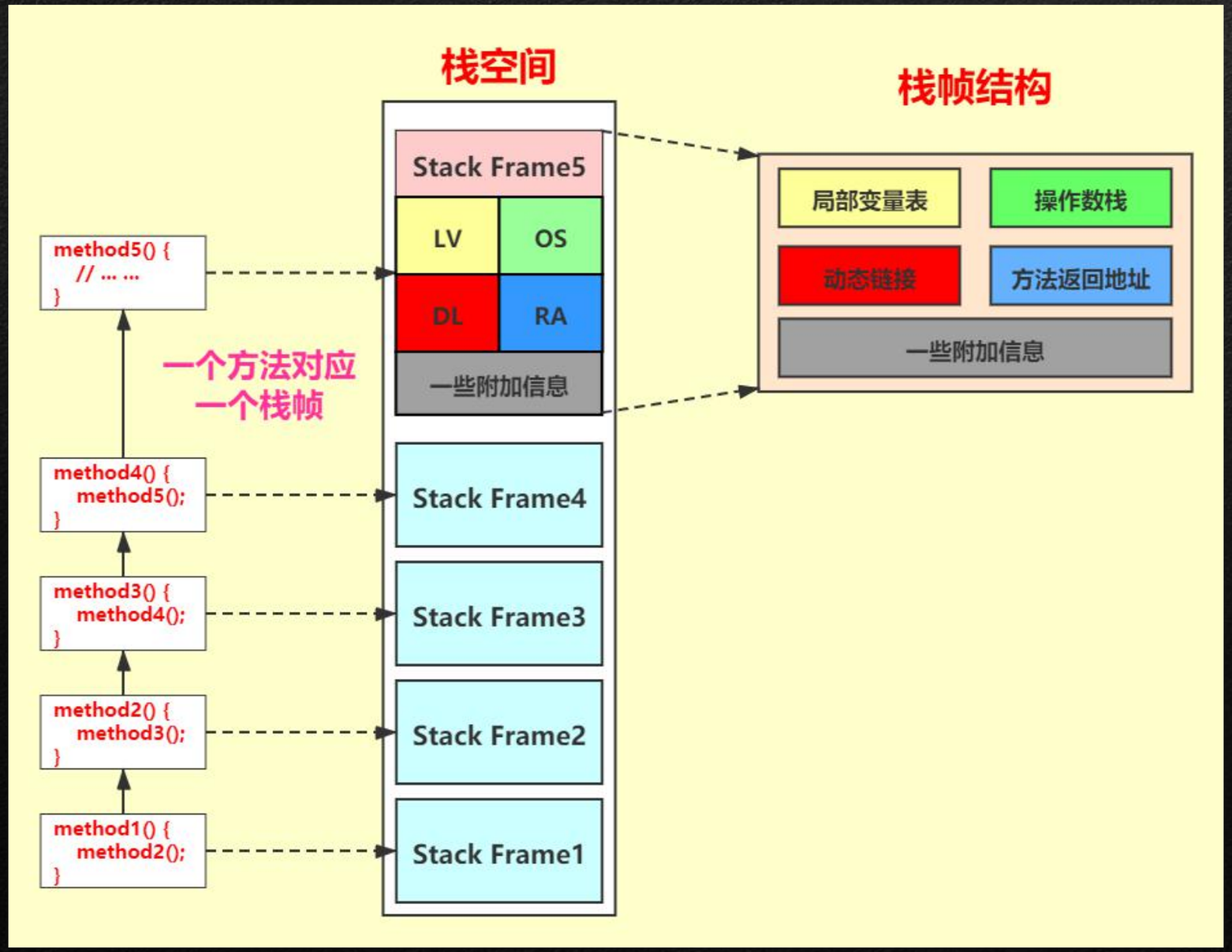
Dalvik虚拟机&ART虚拟机与Hotspot区别

Android应用程序运行在Dalvik/ART虚拟机，并且每一个应用程序对应有一个单独的Dalvik虚拟机实例。Dalvik虚拟机实则也算是一个Java虚拟机，只不过它执行的不是class文件，而是dex文件。

Dalvik虚拟机与Java虚拟机共享有差不多的特性，差别在于两者执行的指令集是不一样的，前者的指令集是基本寄存器的，而后者的指令集是基于堆栈的。

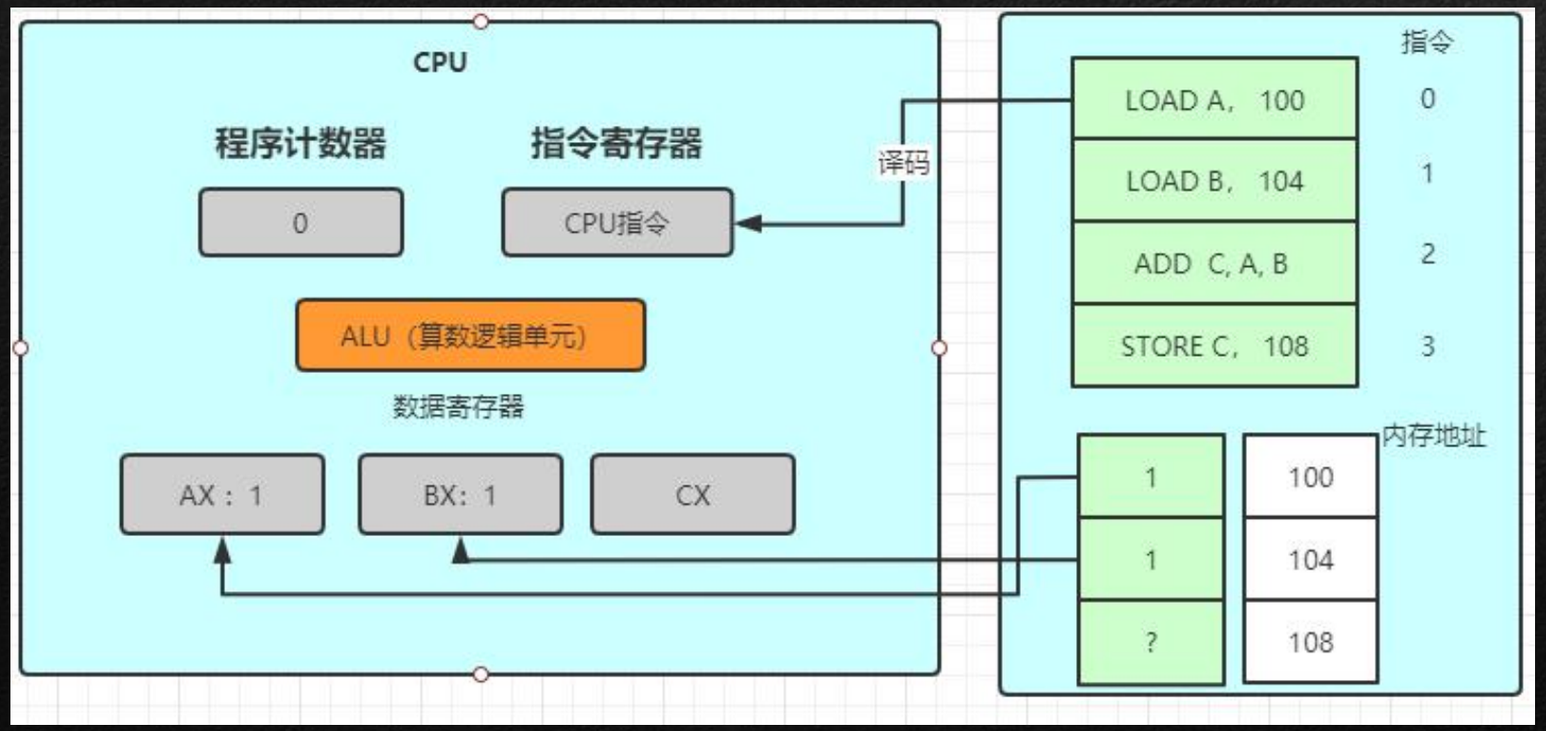
	Java Virtual Machine	Dalvik Virtual Machine
Instruction Set	Java Bytecode (Stack Based)	Dalvik Bytecode (Register Based)
File Format	.class file (one file, one class)	.dex file (one file, many classes)

栈区存储结构与运行原理



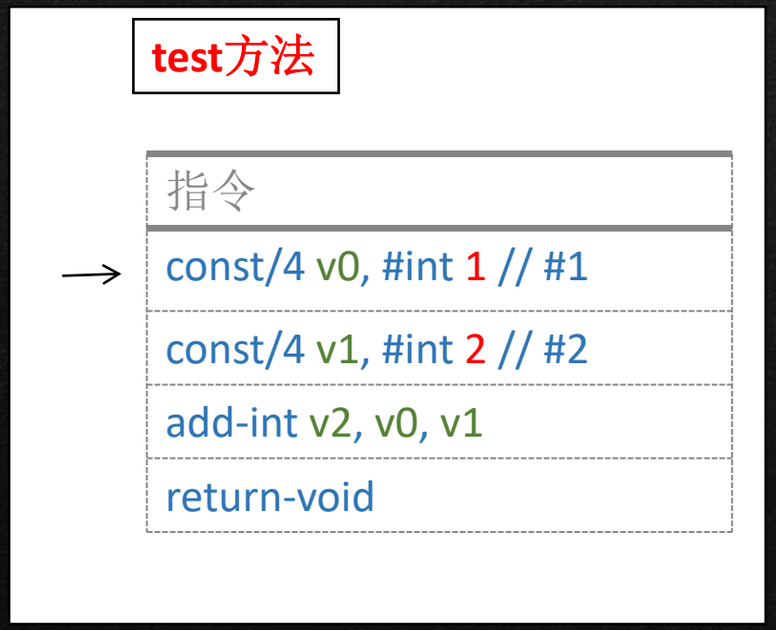
寄存器

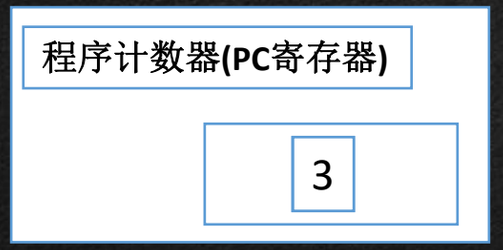
寄存器是CPU的组成部分。寄存器是有限存贮容量的高速存贮部件，它们可用来暂存指令、数据和位址。

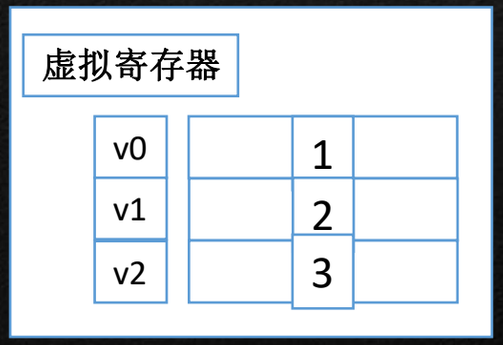


基于寄存器的虚拟机

基于寄存器的虚拟机中没有操作数栈，但是有很多虚拟寄存器。其实和操作数栈相同，这些寄存器也存放在运行时栈中，本质上就是一个数组。与JVM相似，在Dalvik VM中每个线程都有自己的PC和调用栈，方法调用的活动记录以帧为单位保存在调用栈上。



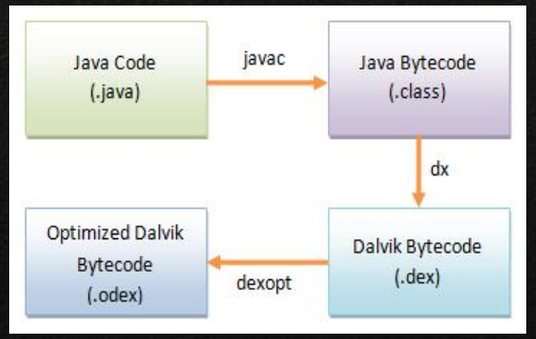




与JVM版相比，可以发现Dalvik版程序的指令数明显减少了，数据移动次数也明显减少了。

Dalvik虚拟机&Hotspot区别

- 基于堆栈的Java指令(1个字节)和基于寄存器的Dalvik指令(2、4或者6个字节)各有优劣
- 一般而言，执行同样的功能， Java虚拟机需要更多的指令（主要是load和store指令），而Dalvik虚拟机需要更多的指令空间
- 需要更多指令意味着要多占用CPU时间，而需要更多指令空间意味着指令缓冲（i-cache）更易失效
- Dalvik虚拟机使用dex（Dalvik Executable）格式的类文件，而Java虚拟机使用class格式的类文件
- 一个dex文件可以包含若干个类，而一个class文件只包括一个类
- 由于一个dex文件可以包含若干个类，因此它可以将各个类中重复的字符串只保存一次，从而节省了空间，适合在内存有限的移动设备使用
- 一般来说，包含有相同类的未压缩dex文件稍小于一个已经压缩的jar文件



ART与Dalvik

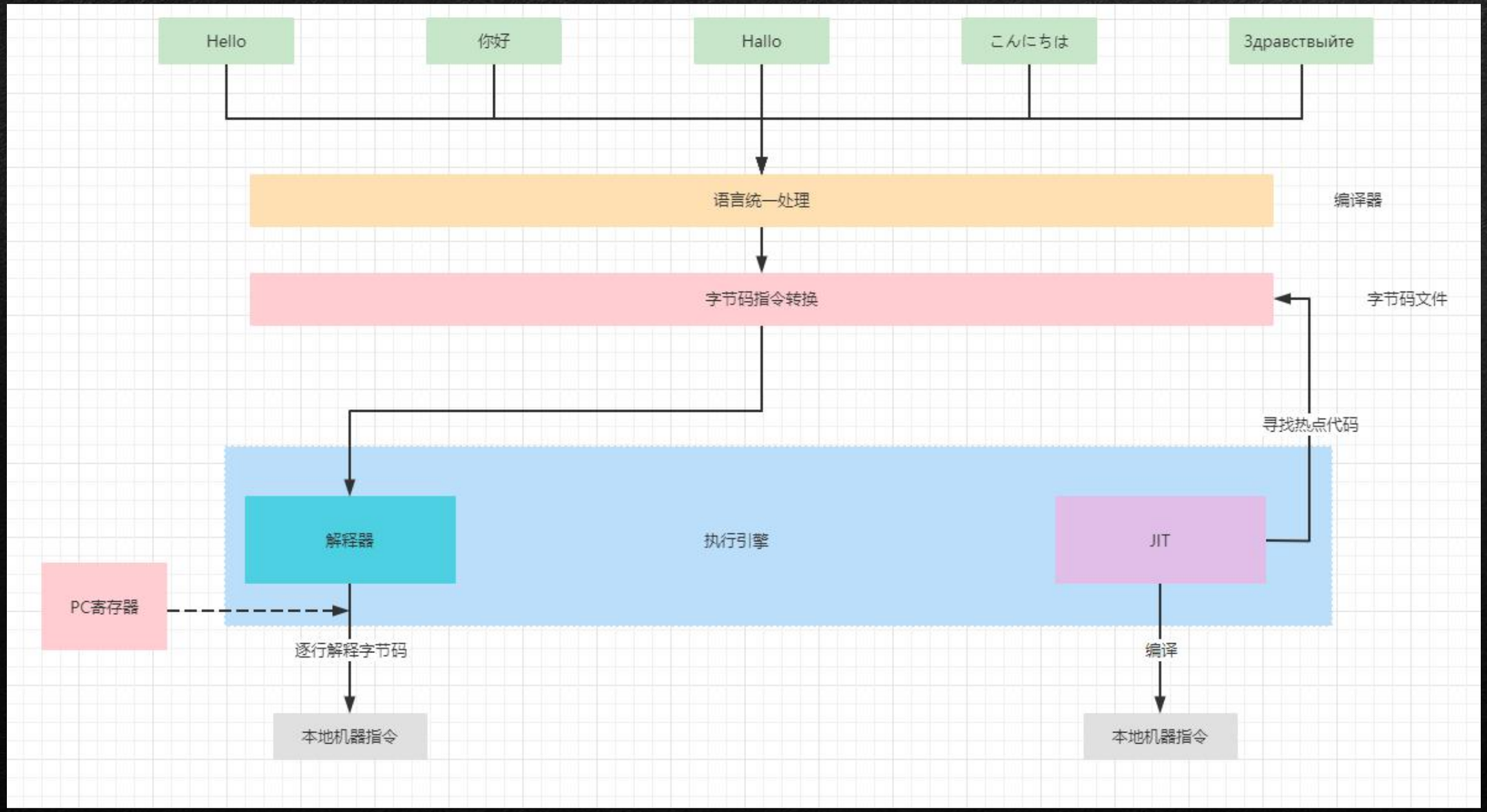
Dalvik虚拟机执行的是dex字节码，解释执行。从Android 2.2版本开始，支持JIT即时编译（Just In Time）

在程序运行的过程中进行选择热点代码（经常执行的代码）进行编译或者优化。

而ART（Android Runtime） 是在 Android 4.4 中引入的一个开发者选项，也是 Android 5.0 及更高版本的默认 Android 运行时。ART虚拟机执行的是本地机器码。Android的运行时从Dalvik虚拟机替换成ART虚拟机，并不要求开发者将自己的应用直接编译成目标机器码，APK仍然是一个包含dex字节码的文件。

那么，ART虚拟机执行的本地机器码是从哪里来？

前端编译器与后端编译器



解释执行&JIT&AOT

解释执行：

程序运行过程中，逐行进行代码编译

JIT：

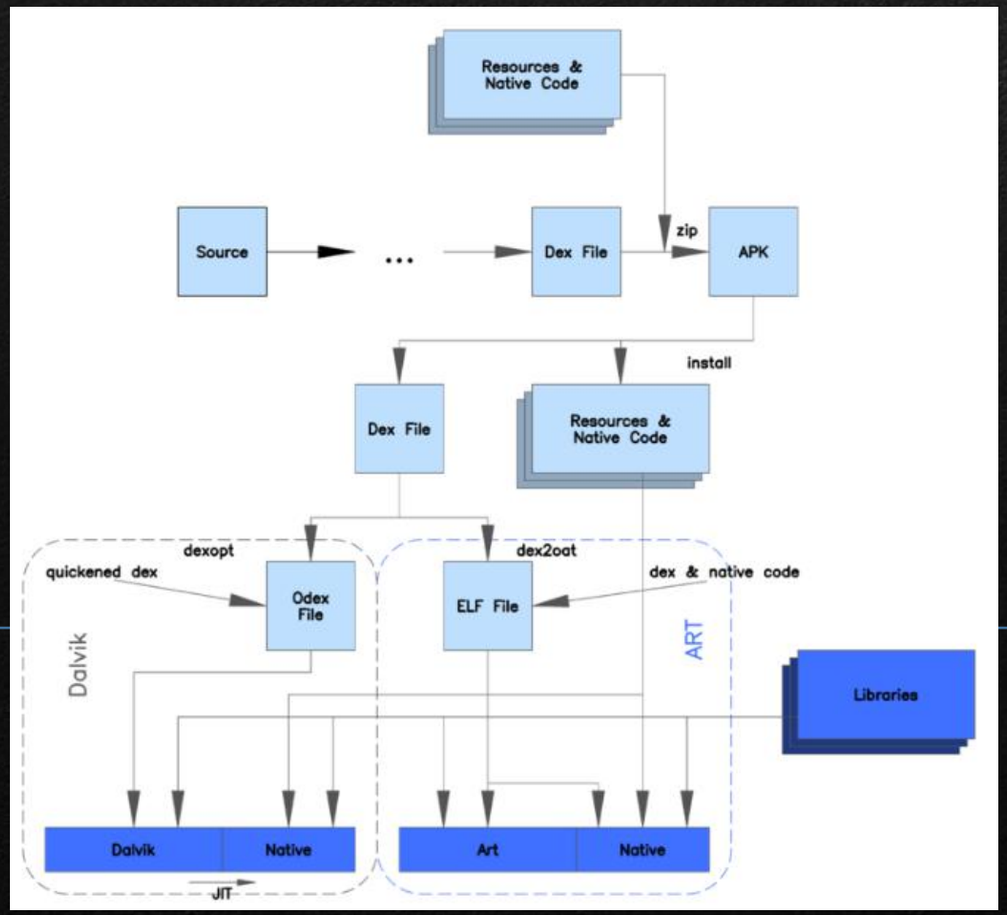
程序运行过程中，将热点代码进行编译缓存执行

AOT：

运行之前，将所有代码打包编译成机器码

dex2aot

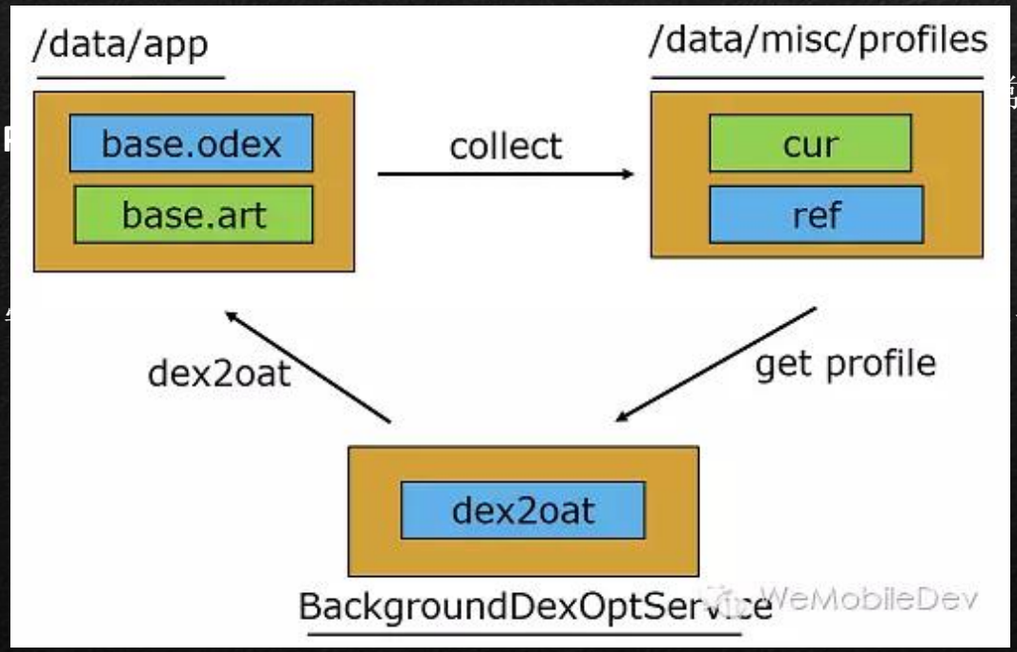
Dalvik下应用在安装的过程，会执行一次优化，将dex字节码进行优化生成odex文件。而Art下将应用的dex字节码翻译成本地机器码的最恰当AOT时机也就发生在应用安装的时候。ART 引入了预先编译机制（Ahead Of Time），在安装时，ART 使用设备自带的 dex2oat 工具来编译应用，dex中的字节码将被编译成本地机器码。



Android N的运作方式

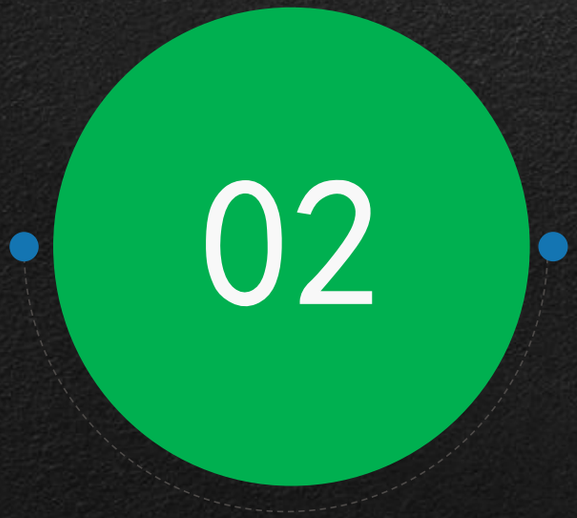
ART 使用预先 (AOT) 编译，并且从 Android N混合使用AOT编译，解释和JIT。

1、最初安装应用时不进行任何
经过 JIT 编译的方法将会记录到



常执行的方法进行JIT，

2、当设备闲置和充电时，编译
译。待下次运行时直接使用。

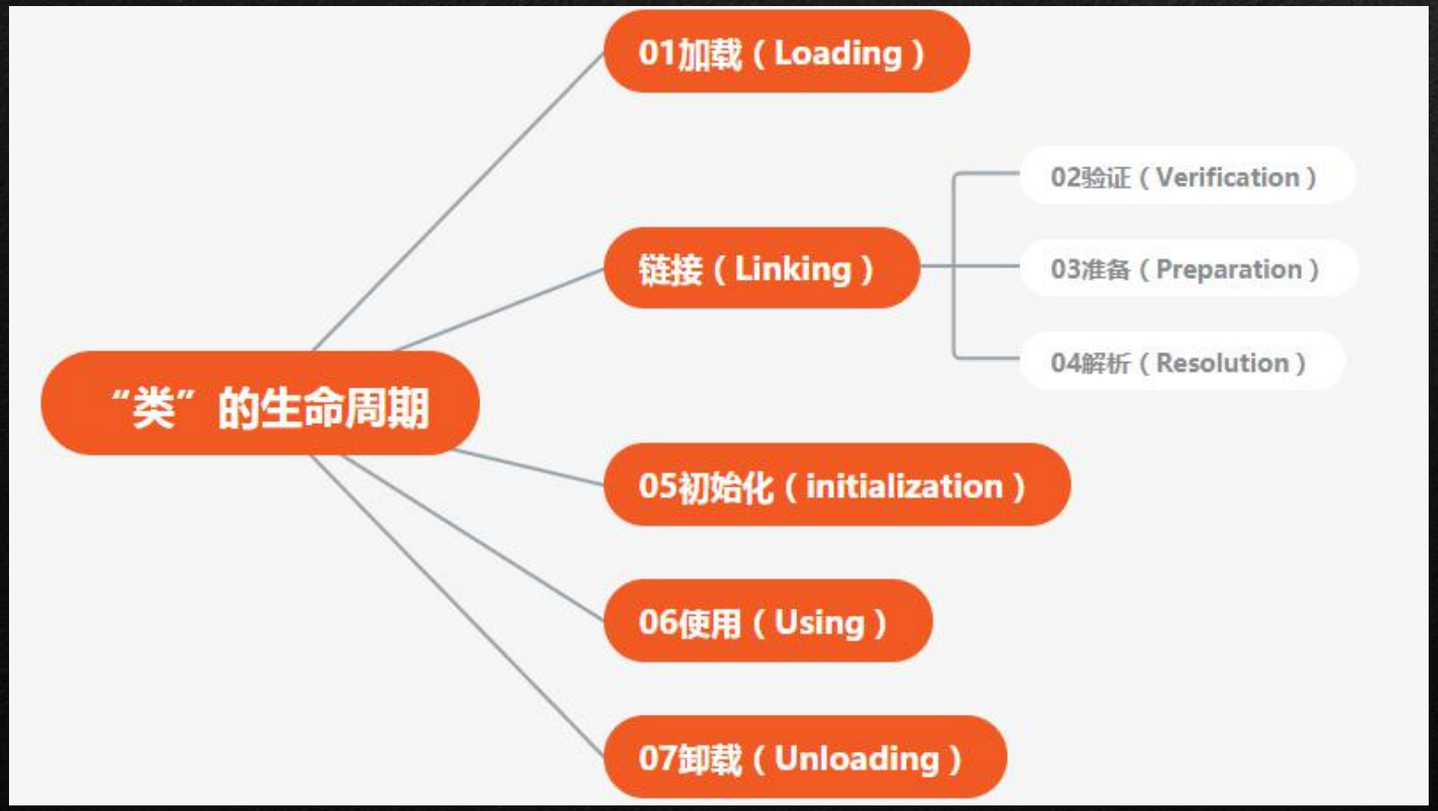


“类（文件）”的生命周期

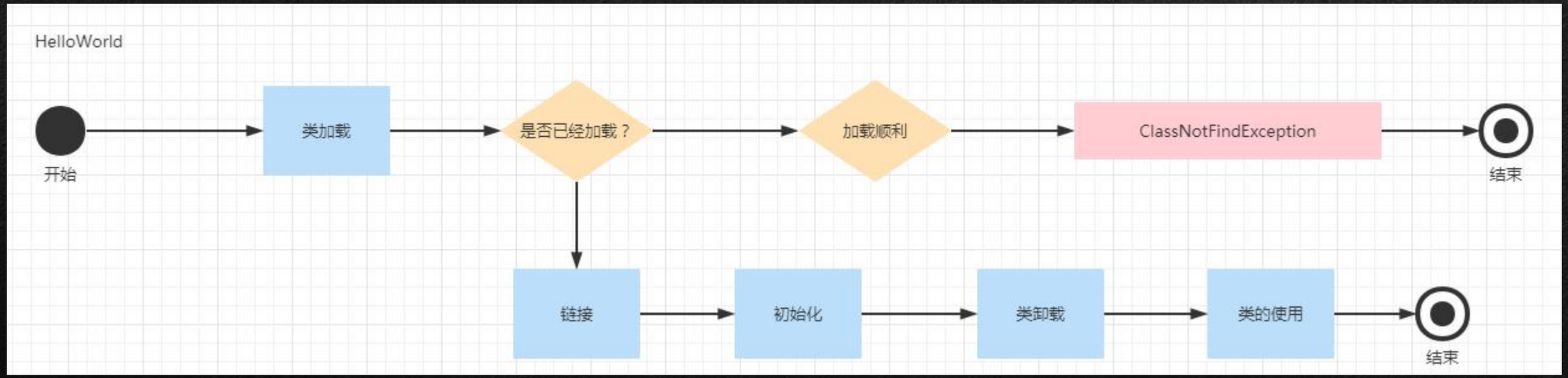
类的生命周期概述

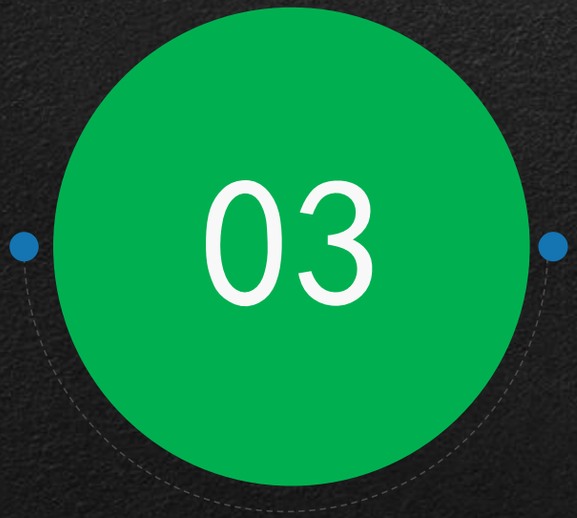
在JAVA中数据类型分为引用数据类型与基本数据类型，基本数据类型由虚拟机预先定义，引用数据类型则需要进行类加载。

按照JAVA虚拟机规范，从class文件到加载到内存当中的类，到类写在出内存位置，他的整个生命周期包含下述七个阶段：



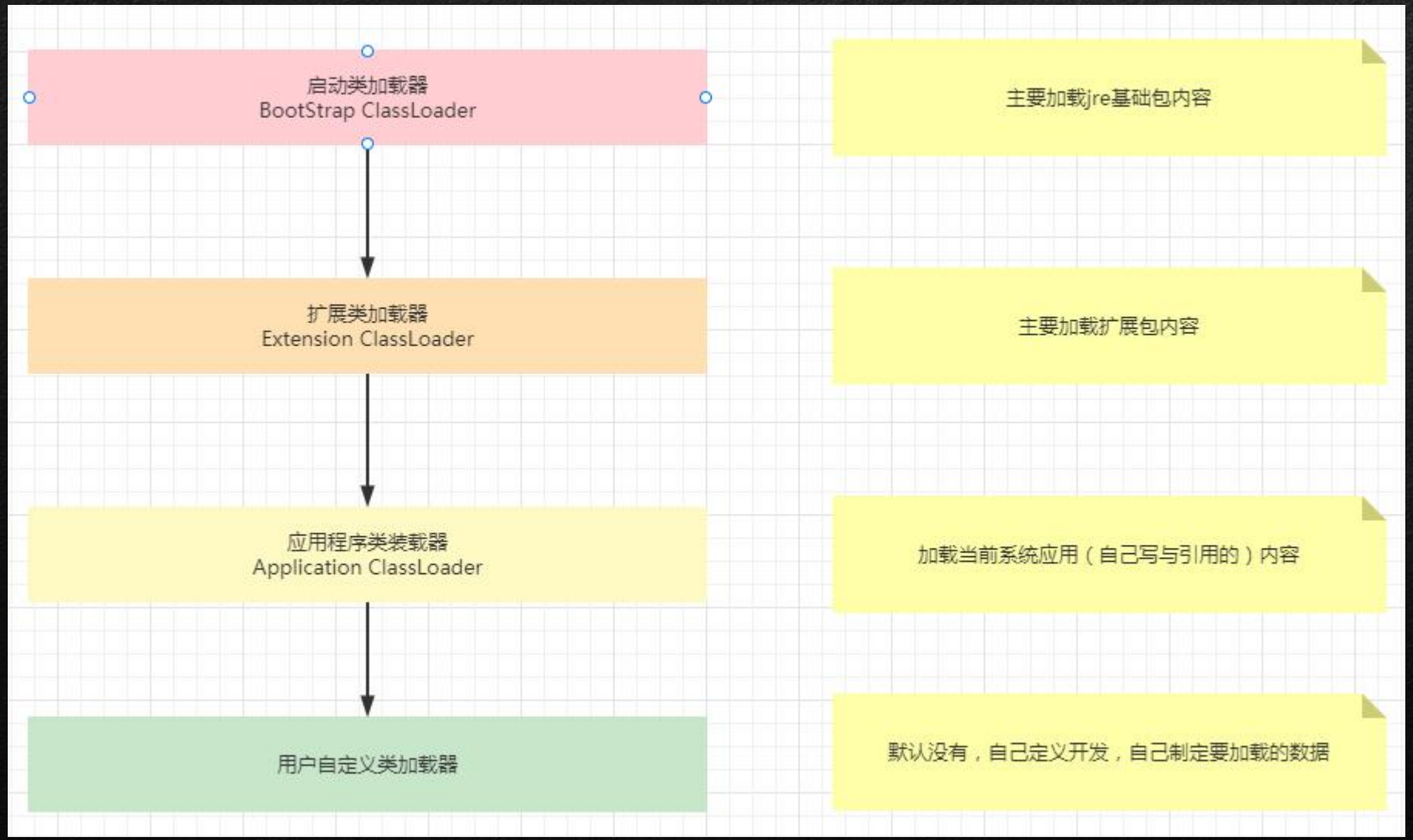
类的使用经过


HelloWorld
开始
类加载
是否已经加载？
加载顺利
ClassNotFindException
结束
链接
初始化
类卸载
类的使用
结束

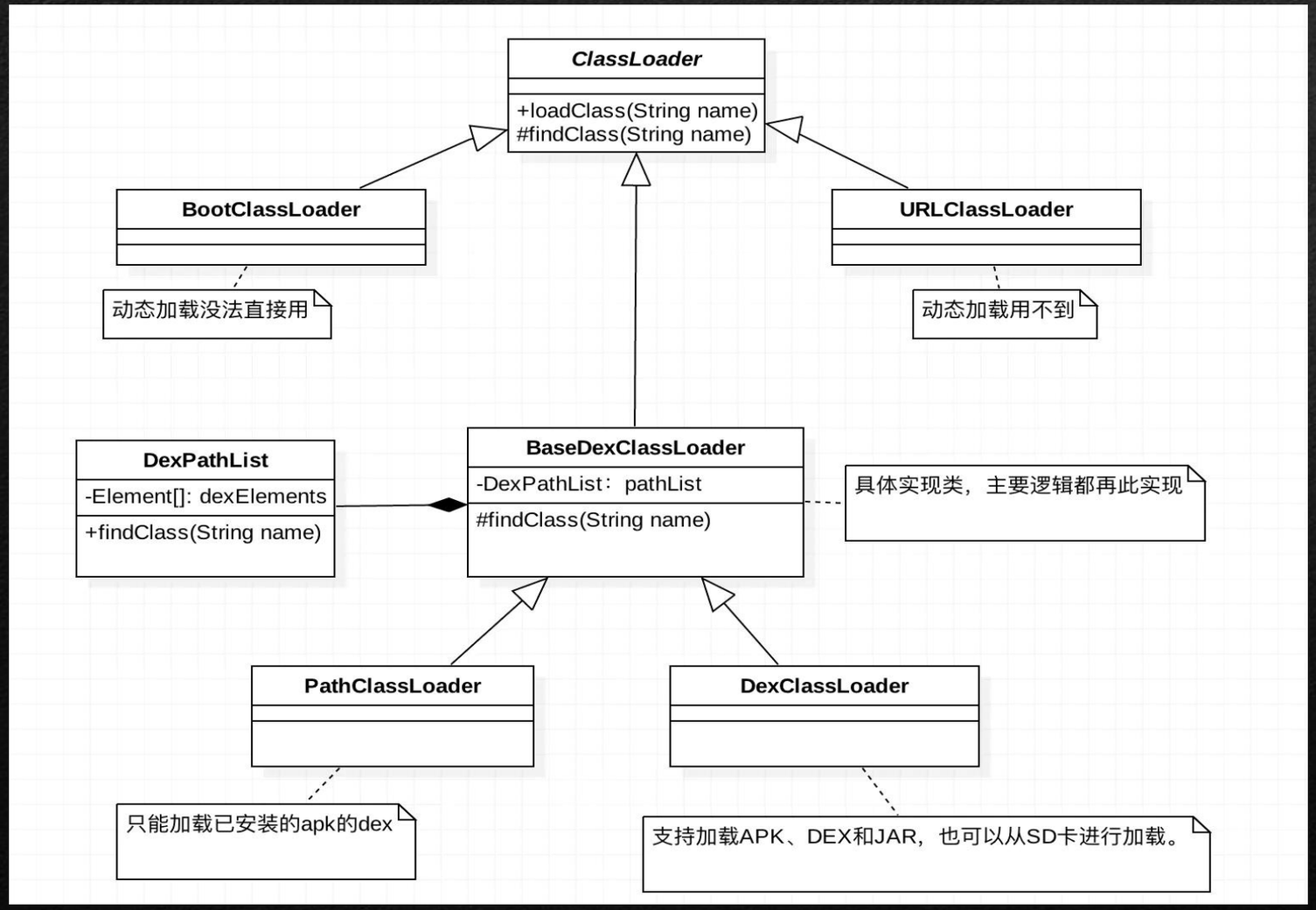


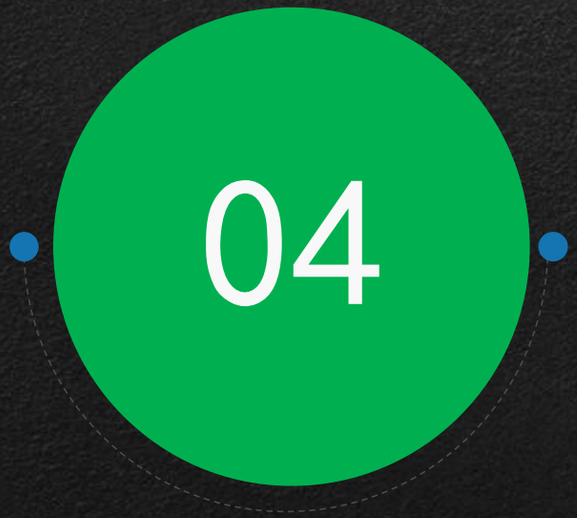
类加载器

类加载器的分类


启动类加载器
BootStrap ClassLoader
主要加载jre基础包内容
扩展类加载器
Extension ClassLoader
主要加载扩展包内容
应用程序类装载器
Application ClassLoader
加载当前系统应用（自己写与引用的）内容
用户自定义类加载器
默认没有，自己定义开发，自己制定要加载的数据

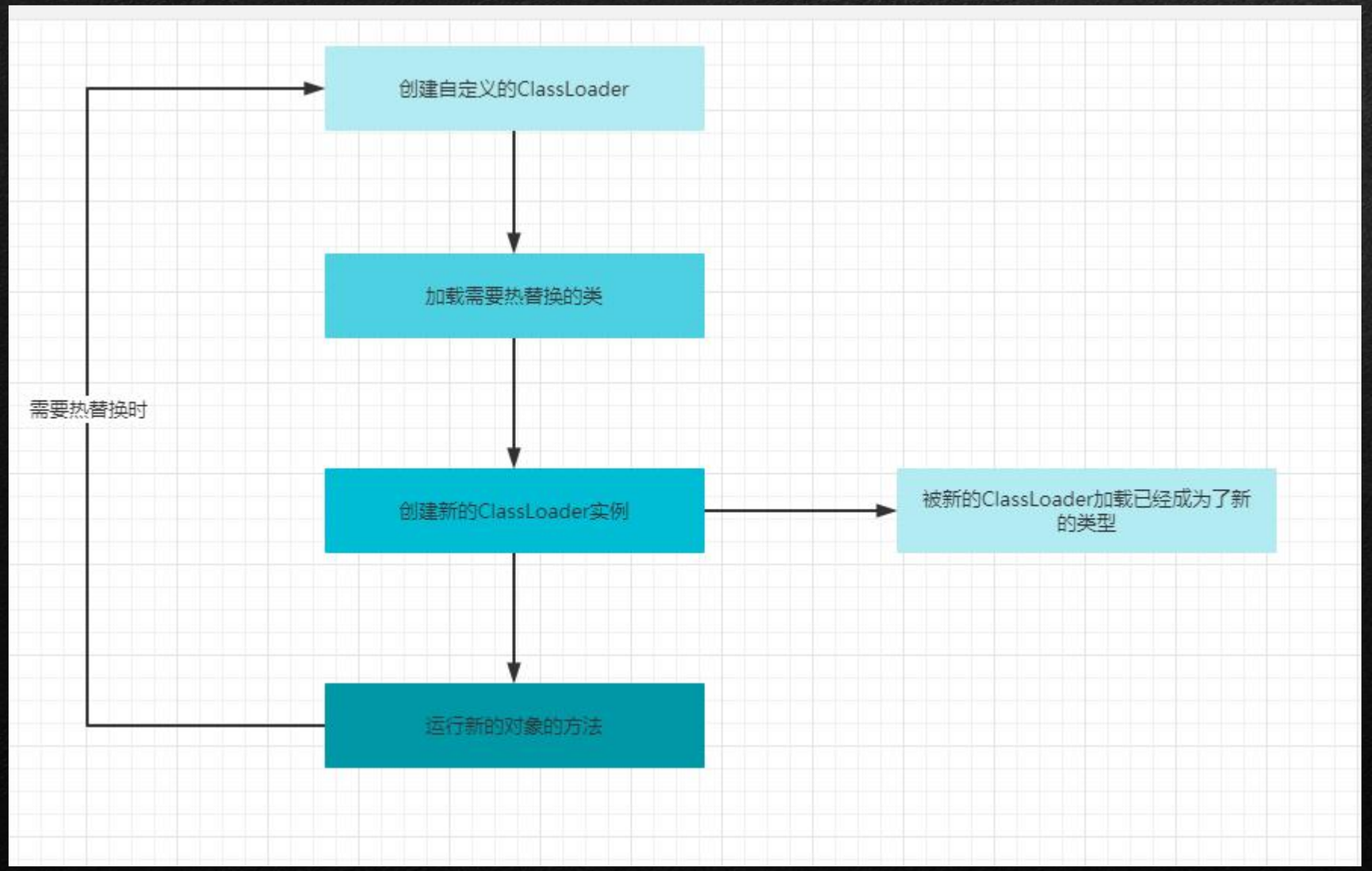
Android类加载器





热修复原理

热修复原理


创建自定义的ClassLoader
加载需要热替换的类
创建新的ClassLoader实例
运行新的对象的方法
需要热替换时
被新的ClassLoader加载已经成为了新的类型

THANK YOU

码牛学院-用代码码出牛逼人生

谢谢观看