

JVM笔记无特殊说明，基于jdk8&14+HotSpot

[零、参考资料 2](#_Toc37923216)

[一、内存结构 3](#_Toc37923217)

[1、程序计数器(Program Counter Register) 3](#_Toc37923218)

[2、Java虚拟机栈(JVM Stack) 3](#_Toc37923219)

[2.2栈帧(Frame) 3](#_Toc37923220)

[2.3局部变量表(Local Variable) 4](#_Toc37923221)

[2.4操作数栈(Operand Stack) 4](#_Toc37923222)

[2.5动态链接(Dynamic Linking) 4](#_Toc37923223)

[2.6方法调用 5](#_Toc37923224)

[2.7 returnAddress数据类型 5](#_Toc37923225)

[2.8 boolean数据类型 5](#_Toc37923226)

[2.9 reference数据类型 5](#_Toc37923227)

[3、本地方法栈(Native Method Stack) 6](#_Toc37923228)

[4、堆(Heap) 7](#_Toc37923229)

[5、方法区(Method Area) 7](#_Toc37923230)

[5.1概述 7](#_Toc37923231)

[5.2运行时常量池(Run-Time Constant Pool) 7](#_Toc37923232)

[6、直接内存(Direct Memory) 7](#_Toc37923233)

[二、编译(Compiling) 8](#_Toc37923234)

[1概述 8](#_Toc37923235)

[2前端编译器 8](#_Toc37923236)

[3即时编译器 8](#_Toc37923237)

[4提前编译器 8](#_Toc37923238)

[5虚拟机指令 8](#_Toc37923239)

[三、类(Class) 9](#_Toc37923240)

[1概述 9](#_Toc37923241)

[2类结构 9](#_Toc37923242)

[2.1类结构元素 9](#_Toc37923243)

[2.2常量池(constant\_pool) 11](#_Toc37923244)

[3类的加载 17](#_Toc37923245)

[3.1类的加载器 17](#_Toc37923246)

[3.2类加载过程 17](#_Toc37923247)

[四、JVM字节码执行 18](#_Toc37923248)

# 零、参考资料

Java虚拟机规范(Java SE 8版)

Java语言规范基于Java SE 8

深入理解Java虚拟机——JVM高级特性与最佳实践

实战Java虚拟机——JVM故障诊断与性能优化

垃圾回收的算法与实现

深入理解计算机系统

Stackoverflow:

https://stackoverflow.com/questions/16549066/java-major-and-minor-garbage-collections

OpenJdk：

https://openjdk.java.net/jeps/122

http://cr.openjdk.java.net/~sundar/8022483/webrev.01/raw\_files/new/src/share/classes/com/sun/tools/hat/resources/oqlhelp.html

Oracle官方DOC：

https://docs.oracle.com/javase/8/javase-books.htm

https://docs.oracle.com/en/java/javase/13/

https://docs.oracle.com/en/java/javase/14/

https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf

https://docs.oracle.com/javase/specs/jvms/se14/jvms14.pdf

https://docs.oracle.com/javase/specs/index.html

https://www.oracle.com/webfolder/technetwork/tutorials/mooc/JVM\_Troubleshooting/week1/lesson1.pdf

https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

资料整理说明：

1. 参考《深入理解Java虚拟机——JVM高级特性与最佳实践》第三版，中文
2. Java虚拟机规范(Java SE 8版)，中文
3. https://docs.oracle.com/javase/specs/jvms/se14/jvms14.pdf，英文
4. jdk14是编写此文档时最新的jdk版本，
5. java虚拟机规范，jdk8和jdk14大致相同，但jdk8之后增加了很多新特性，所以主要参考jdk14规范

# 一、内存结构

A screenshot of a cell phone

Description automatically generated

PS：蓝色块为线程独享空间

## 1、程序计数器(Program Counter Register)

1.1概述

a当前线程正在执行Java方法，记录所执行的虚拟机指令的地址。否则为空(Undefined)。

b每个线程都有独立的PC，

c字节码解释器工作时通过改变PC的值来选取下一条需要执行的字节码指令

d任意时刻JVM线程只会执行一个方法的代码(At any point, each Java Virtual Machine thread is executing the code of a single method)。

e这块的容量至少保存一个returnAddress类型的数据或一个与平台相关的本地指针的值(wide enough to hold a returnAddress or a native pointer on the specific platform)

f唯一一个在规范中没有规定任何OOM的区域

## 2、Java虚拟机栈(JVM Stack)

2.1概述

a每线程都有独立的虚拟机栈，生命周期和线程相同，可以在堆中分配

b分配的内存地址可以不需要保证连续

c每个方法执行时，都会同步创建个栈帧(Frame)

d栈的大小，可以固定，也可以配置动态收缩。

e栈的大小为固定时，线程可在创建时自定义栈大小

new Thread(ThreadGroup group, Runnable target, String name, long stackSize)

f会抛出StackOverflowError和OutOfMemoryError

### 2.2栈帧(Frame)

2.2.1概述

a用于存储数据和部分过程结果的数据结构

b用于处理动态链接(Dynamic Linking),方法返回值(return values for methods)和异常分派(dispatch exceptions)

c生命周期同方法调用过程，且无论方法是否有异常，均视为结束，并销毁

(A frame is destroyed when its method invocation completes, whether that completion is normal or abrupt (it throws an uncaught exception))

d每个栈帧都有自己的一组局部变量表(Local Variable)、操作数栈(Operand Stack)、和指向当前方法所属类的运行时常量池的引用(a reference to the runtime constant pool of the class of the current method)

e栈帧数据结构大小仅取决于虚拟机的实现

d栈帧是线程私有数据

### 2.3局部变量表(Local Variable)

2.3.1概述

a长度由编译期决定，并通过方法的code属性(这里)保存以及提供给栈帧使用(supplied in the binary representation of a class or interface along with the code for the method associated with the frame)

b一个局部变量槽(Slot)可以保存一个类型为 boolean、byte、char、short、int、float、对象引用(reference)、 returnAddress的数据，long或者double需要两个连续的局部变量槽

c局部变量通过索引(indexing)来定位访问，索引从0开始

d一个局部变量槽的大小，由具体的虚拟机设计决定

e无论读写变量槽是否为原子操作，都不会引起数据竞争和线程安全问题。

f变量槽可以复用

### 2.4操作数栈(Operand Stack)

2.4.1概述

a后进先出(last-in-first-out, LIFO)

b栈深度在编译期决定，并通过方法的code属性保存以及提供给栈帧使用

c全称是：当前栈帧的操作数栈(Where it is clear by context, we will sometimes refer to the operand stack of the current frame as simply the operand stack.)

d在栈帧创建时，操作数栈是空的

e栈中元素的数据类型必须与字节码指令的序列严格匹配

f JVM的解释执行器引擎被称为“基于栈的执行引擎”，这个栈就是，操作数栈

### 2.5动态链接(Dynamic Linking)

2.5.1概述

a通过一个指向当前方法所在类型的运行时常量池的引用，对当前方法的代码实现动态链接(Each frame contains a reference to the run-time constant pool for the type of the current method to support dynamic linking of the method code)

b类文件的常量池中存有大量的符号引用，字节码中的方法调用指令就是以常量池指向方法的符号引用作为参数

b.1静态解析：指一部分符号在类加载阶段或者第一次使用的时候就被转化为直接引用的转化

b.2动态链接：指一部分符号在每一次运行期间转化为直接引用的转化

### 2.6方法调用

2.6.1概述

a正常调用完成是指在方法执行过程中没有抛出任何异常，包括但不限于通过throw语句显式抛出的异常(A method invocation completes normally if that invocation does not cause an exception to be thrown, either directly from the Java Virtual Machine or as a result of executing an explicit throw statement)

b如果有返回值的话，使用哪种返回指令取决于方法返回值的数据类型

c异常调用完成是指方法在执行过程中抛出了异常，并且没有捕获异常

d方法异常调用完成，一定不会有方法返回值返回给调用者(A method invocation that completes abruptly never returns a value to its invoker)

e方法退出后都必须返回到最初方法被调用的位置，程序才能继续执行

f方法退出过程实际上等同于吧当前栈帧出栈

### 2.7 returnAddress数据类型

a用于jsr,ret,jsr\_w 三种虚拟机指令

b值指向一个jvm操作指令，

cjava语言中没有对应的类型，且无法在运行中改变

d在旧的jvm中用于异常处理，现在用异常表代替

### 2.8 boolean数据类型

a JVM对boolean支持很有限，没有单独的操作指令

b对于java中boolean类型数据，会被编译成jvm中int数据类型(*Instead, expressions in the Java programming language that operate on boolean values are compiled to use values of the Java Virtual Machine int data type*)

c对于boolean数组,JVM复用byte数组的baload和bastore指令

d JVM用1表示true，0表示false

### 2.9 reference数据类型

a三种引用类型：类类型(class Types)、数组类型、接口类型

b默认值是null，但虚拟机没规定null值怎么表示(*The default value of a reference type is null. This specification does not mandate a concrete value encoding null*)

c没有规定reference的长度和结构，

d一般来说要做到下面两件事：

d.1可以直接或间接的查找到对象在堆中的数据存放的起始地址

d.2可以直接或间接的查找到对象所属数据类型在方法区中的存储的类型信息

e如果引用时数组类型，最外面那一维元素的类型，称为该数组的组件类型(component type)

*An array type consists of a component type with a single dimension (whose length is not given by the type)*

f元素类型(element type)定义太拗口，参考虚拟机规范jdk14的2.4章节

## 3、本地方法栈(Native Method Stack)

3.1概述

a为虚拟机用到的本地(Native)方法服务

b如果虚拟机不支持Native方法或不依赖传统栈(通常称C stack)，可以没有本地方法栈

c会抛出StackOverflowError和OutOfMemoryError

e栈的大小，可以固定，也可以配置动态收缩。

f栈的大小为固定时，线程可在创建时自定义栈大小

e虚拟机栈和本地方法栈可以被设计为合二为一，比如HotSpot虚拟机。

## 4、堆(Heap)

4.1概述

a在虚拟机启动时创建，被所有线程共享

b可以划分出线程私有的线程缓冲区(Thread Local Allocation Buffer,TLAB)

c内存地址不需要保证连续，但在逻辑上应被视为连续的

d部分虚拟机设计，会针对大对象(比如数组)存储，要求内存地址是连续的

e容量可以固定，也可以动态扩展

f存储的对象无法显式销毁(objects are never explicitly deallocated)

g会抛出OutOfMemoryError

## 5、方法区(Method Area)

### 5.1概述

a别名非堆(Non-heap),不等价于永久代(Permanent Generation),被所有线程共享

b从JDK8开始，在本地内存中实现的元空间(Metaspace)代替永久代实现方法区

c存储了每个类的结构信息(It stores per-class structures),比如，运行时常量池(the run-time constant pool)，字段和方法数据(field and method data)，构造函数和普通方法的字节码内容(the code for methods and constructors)，还包括一些在类、实例、接口初始化时用到的特殊方法(including the special methods used in class and instance initialization and interface initialization.)

d容量可以固定，也可以动态扩展，可以不实现GC

e会抛出OutOfMemoryError

f此区域的数据存储结构由JVM自定义，JVM规范不做具体规定

### 5.2运行时常量池(Run-Time Constant Pool)

2.1概述

a是class文件中每个类或接口的常量池表(constant\_pool)的运行时表现形式(a per-class or per-interface run-time representation of the constant\_pool table in a class file)

b包含编译期已知的数值字面量和运行期解析后获取的方法或字段引用(ranging from numeric literals known at compile-time to method and fi eld references that must be resolved at run-time)

ps:String.intern()

c会抛出OutOfMemoryError

d虚拟机规范没有对此作任何细节的要求

## 6、直接内存(Direct Memory)

6.1概述

a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

b可能会抛出OutOfMemoryError

c可以通过DirectByteBuffer对象作为这块内存的引用进行操作

# 二、编译(Compiling)

## 1概述

1.1这块虚拟机规范中只做示意，并不属于规范内容

1.2

## 2前端编译器

## 3即时编译器

## 4提前编译器

## 5虚拟机指令

# 三、类(Class)

## 1概述

1.1类文件格式不是必须存在于文件里。

1.2多字节数据项必须严格按照大端在前(big-endian)顺序进行存储(Multibyte data items are always stored in big-endian order,where the high bytes come first)

1.3 Class文件是一组以8字节为基础单位的二进制流，各个数据项严格按照顺序紧凑排列，没有任何分隔符

1.4Class文件格式采用一种类似C语言结构体的伪结构(This chapter presents the class file format using pseudostructures written in a C-like structure notation)，只有两种数据类型:无符号数(unsigned)和表(table)

1.5无符号数是基本的数据类型，可以用来描述数字、索引引用、数量值或者按照UTF-8编码构成字符串值，以u1,u2,u4,u8来分别代表1个字节，2个字节，4个字节，8个字节

1.6表由多个无符号数或其他表作为数据项构成的复合数据类型

1.7当描述同一类型但数量不定的多个数据时，经常会使用一个前置的容量计数器加若干个连续的数据项的形式，这时候称这一系列连续的某一类型的数据为某一类型的集合

## 2类结构

### 2.1类结构元素

|  |  |  |
| --- | --- | --- |
| 类型 | 数量 | 说明 |
| u4 | 1 | 魔数(magic)，确定是否为虚拟机所接受的类文件，固定值为0xCAFEBABE |
| u2 | 1 | 副版本号(minor\_version)  1、主版本号为45-55时，副版本号可以为任何值  2、从主版本号56(jdk12)开始，副版本号必须为0或65535  3、当副版本号为65535时，为预览版 |
| u2 | 1 | 主版本号(major\_version)，  1、值从45开始  2、即使文件格式无任何变化，虚拟机也必须拒绝执行类主版本号超过虚拟机版本号的文件。 |
| u2 | 1 | constant\_pool\_count;常量池计数器 |
| cp\_info | constant\_pool[constant\_pool\_count-1] | 常量池  1、是一种表结构  2、以1～constant\_pool\_count-1为索引  3、第一个字节作为类型标记，这个字节称为标签字节(tag byte)  4、主要存放两大类常量：字面量(Literal)和符号引用(Symbolic References)  5、通用格式：  cp\_info {  u1 tag;  u1 info[];  } |
| u2 | 1 | access\_flags;访问标志，  1、一种由标志构成的掩码  The value of the access\_flags item is a mask of flags used to denote access permissions to and properties of this class or interface  2、取值以及含义参考2.3  3、没有使用到的标志位都为0 |
| u2 | 1 | this\_class;类索引  1、取值为常量池有效的索引值  2、常量池中索引处的成员必须为CONSTANT\_Class\_info类型 |
| u2 | 1 | super\_class;父类索引  1、Java不允许多重继承，所以父类索引只有一个  2、取值要么为0，要么是常量池中某个有效索引值  3、当值为0，此类只能用于表示Object类  4、当值不为0，常量池中索引处的成员必须为CONSTANT\_Class\_info类型  5、所有父类的access\_flags，均不能带有ACC\_FINAL标志  Neither the direct superclass nor any of its superclasses may have the ACC\_FINAL flag set in the access\_flags item of its ClassFile structure. |
| u2 | 1 | interfaces\_count;接口索引计数器，如果没有接口，则取值0 |
| u2 | interfaces[interfaces\_count] | 接口集合  1、每个元素值都是常量池中某个有效索引值  2、按implements关键字后面的接口，从左到右排列 |
| u2 | 1 | fields\_count;字段计数器 |
| field\_info | fields[fields\_count] | 字段表  1、每个成员都是field\_info结构，参考2.4  2、包括类变量和实例变量  3、不包括方法内声明的局部变量  4、不包括从父类或父接口继承的那些字段 |
| u2 | 1 | methods\_count;方法计数器 |
| method\_info | methods[methods\_count] | 1、每个成员都是method\_info结构，参考2.5  2、方法表只描述当前类或接口中声明的方法，不包括从父类或父接口继承的方法  3、支持方法访问权限(access\_flags)既不是ACC\_NATIVE又不是ACC\_ABSTRACT |
| u2 | 1 | attributes\_count;属性计数器 |
| attribute\_info | attributes[attributes\_count] | 1、每个成员都是attribute\_info结构，参考2.6 |

### 2.2常量池(constant\_pool)

#### 2.2.1基本格式

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| tag | u1 |  |
| info[] | u1 |  |

#### 2.2.2常量池类型(tag)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | tag | 类文件版本 | 首次javaSE版本 | 说明 |
| CONSTANT\_Utf8 | 1 | 45.3 | 1.0.2 | UTF-8编码的字符串 |
| CONSTANT\_Integer | 3 | 45.3 | 1.0.2 | 整型字面量 |
| CONSTANT\_Float | 4 | 45.3 | 1.0.2 | 浮点型字面量 |
| CONSTANT\_Long | 5 | 45.3 | 1.0.2 | 长整型字面量 |
| CONSTANT\_Double | 6 | 45.3 | 1.0.2 | 双精度浮点型字面量 |
| CONSTANT\_Class | 7 | 45.3 | 1.0.2 | 类或接口的索引 |
| CONSTANT\_String | 8 | 45.3 | 1.0.2 | 字符串类型字面量 |
| CONSTANT\_Fieldref | 9 | 45.3 | 1.0.2 | 字段的符号引用 |
| CONSTANT\_Methodref | 10 | 45.3 | 1.0.2 | 类中方法的符号引用 |
| CONSTANT\_InterfaceMethodref | 11 | 45.3 | 1.0.2 | 接口中方法的符号引用 |
| CONSTANT\_NameAndType | 12 | 45.3 | 1.0.2 | 字段或方法的部分符号引用 |
| CONSTANT\_MethodHandle | 15 | 51 | 7 | 表示方法具柄 |
| CONSTANT\_MethodType | 16 | 51 | 7 | 方法类型 |
| CONSTANT\_InvokeDynamic | 18 | 51 | 7 | 表示一个动态方法调用点 |
| CONSTANT\_Module | 19 | 53 | 9 | 表示一个模块 |
| CONSTANT\_Package | 20 | 53 | 9 | 表示一个模块中开放或者导出的包 |
| CONSTANT\_Dynamic | 17 | 55 | 11 | 表示一个动态计算常量 |

2.2.2常量池数据类型的结构

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 字段 | 字段类型 | 说明 |
| CONSTANT\_Utf8 | tag | u1 | UTF-8编码的字符串 |
| length | u2 |
| bytes[] | u1 |
| CONSTANT\_Integer | 3 | 45.3 | 整型字面量 |
|  |  |
| CONSTANT\_Float | 4 | 45.3 | 浮点型字面量 |
| CONSTANT\_Long | 5 | 45.3 | 长整型字面量 |
| CONSTANT\_Double | 6 | 45.3 | 双精度浮点型字面量 |
| CONSTANT\_Class | 7 | 45.3 | 类或接口的索引 |
| CONSTANT\_String | 8 | 45.3 | 字符串类型字面量 |
| CONSTANT\_Fieldref | 9 | 45.3 | 字段的符号引用 |
| CONSTANT\_Methodref | 10 | 45.3 | 类中方法的符号引用 |
| CONSTANT\_InterfaceMethodref | 11 | 45.3 | 接口中方法的符号引用 |
| CONSTANT\_NameAndType | 12 | 45.3 | 字段或方法的部分符号引用 |
| CONSTANT\_MethodHandle | 15 | 51 | 表示方法具柄 |
| CONSTANT\_MethodType | 16 | 51 | 方法类型 |
| CONSTANT\_InvokeDynamic | 18 | 51 | 表示一个动态方法调用点 |
| CONSTANT\_Module | 19 | 53 | 表示一个模块 |
| CONSTANT\_Package | 20 | 53 | 表示一个模块中开放或者导出的包 |
| CONSTANT\_Dynamic | 17 | 55 | 表示一个动态计算常量 |

2.3访问标志(access\_flags)

|  |  |  |
| --- | --- | --- |
| 标志 | 值 | 说明 |
| ACC\_PUBLIC | 0x0001 | public |
| ACC\_FINAL | 0x0010 | final，不能与ACC\_ABSTRACT同时设置 |
| ACC\_SUPER | 0x0020 | 是否使用invokespecial指令的新语义，因为invokespecial指令在jdk1.0.2以后做了改变，所以在jdk1.0.2之后，这里都为真 |
| ACC\_INTERFACE | 0x0200 | 接口，设置了此值，ACC\_ABSTRACT必须同时设置  ACC\_FINAL, ACC\_SUPER, ACC\_ENUM和 ACC\_MODULE，必须不能设置 |
| ACC\_ABSTRACT | 0x0400 | 抽象类，不能实例化 |
| ACC\_SYNTHETIC | 0x1000 | 标志该类或接口是由编译器生成 |
| ACC\_ANNOTATION | 0x2000 | 声明为注解，同时必须设置ACC\_SYNTHETIC |
| ACC\_ENUM | 0x4000 | 声明为枚举 |
| ACC\_MODULE | 0x8000 | 声明为模块，如果这个值被设置，则不能有其他标志被设置  If the ACC\_MODULE flag is set in the access\_flags item, then no other flag in the access\_flags item may be set |

2.4字段表结构(field\_info)

2.4.1基础结构

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| access\_flags | u2 | 访问标志，参考2.4.2 |
| name\_index | u2 | 字段简单名称，指向常量池一个CONSTANT\_Utf8\_info值的索引 |
| descriptor\_index | u2 | 字段的描述符，指向常量池一个CONSTANT\_Utf8\_info值的索引 |
| attributes\_count | u2 | 参考2.7 |
| attributes[attributes\_count] | attribute\_info | 参考2.7 |

2.4.2字段访问标志

|  |  |  |
| --- | --- | --- |
| 标志 | 值 | 说明 |
| ACC\_PUBLIC | 0x0001 | public |
| ACC\_PRIVATE | 0x0002 | private |
| ACC\_PROTECTED | 0x0004 | protected |
| ACC\_STATIC | 0x0008 | static |
| ACC\_FINAL | 0x0010 | final |
| ACC\_VOLATILE | 0x0040 | volatile |
| ACC\_TRANSIENT | 0x0080 | transient |
| ACC\_SYNTHETIC | 0x1000 | 标志该字段是由编译器生成，如果字段存在这个标志，则不能有其他任何访问标志 |
| ACC\_ENUM | 0x0001 | 枚举 |

a ACC\_PUBLIC、ACC\_PRIVATE、ACC\_PROTECTED三选一

b ACC\_FINAL、ACC\_VOLATILE二选一

c接口中的字段必须有ACC\_PUBLIC,ACC\_STATIC,ACC\_FINAL，三个标志

2.5方法结构(method\_info)

2.5.1基础结构

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| access\_flags | u2 | 访问标志，参考2.5.2 |
| name\_index | u2 | 字段简单名称，指向常量池一个CONSTANT\_Utf8\_info值的索引，必须是有效的非限定名称，或者<init>，或者<clinit> |
| descriptor\_index | u2 | 字段的描述符，指向常量池一个CONSTANT\_Utf8\_info值的索引，  1、当是类的<init>方法，必须描述为void方法  2、当类名是<clinit>，必须描述为void方法，且jdk7以后，此方法无参数 |
| attributes\_count | u2 | 参考2.7 |
| attributes[attributes\_count] | attribute\_info | 参考2.7 |

2.5.2访问标志

|  |  |  |
| --- | --- | --- |
| 标志 | 值 | 说明 |
| ACC\_PUBLIC | 0x0001 | public |
| ACC\_PRIVATE | 0x0002 | private |
| ACC\_PROTECTED | 0x0004 | protected |
| ACC\_STATIC | 0x0008 | static，jdk7以后<clinit>方法必须有此标志 |
| ACC\_FINAL | 0x0010 | final |
| ACC\_SYNCHRONIZED | 0x0020 | synchronized |
| ACC\_BRIDGE | 0x0040 | 桥接方法，为了泛型方法兼容jdk1.5以前版本，编译器自动生成 |
| ACC\_VARARGS | 0x0080 | 有可变长参数的标志 |
| ACC\_NATIVE | 0x0100 | native |
| ACC\_ABSTRACT | 0x0400 | abstract |
| ACC\_STRICT | 0x0800 |  |
| ACC\_SYNTHETIC | 0x1000 |  |

a ACC\_PUBLIC, ACC\_PRIVATE,ACC\_PROTECTED 必须三选一

b jdk8以前，接口的方法必须有ACC\_PUBLIC 和ACC\_ABSTRACT标志

c jdk8以及以后，接口的方法必须ACC\_PUBLIC ， ACC\_PRIVATE二选一

d 方法如果标记了ACC\_ABSTRACT，则不能有ACC\_PRIVATE, ACC\_STATIC, ACC\_FINAL,ACC\_SYNCHRONIZED, ACC\_NATIVE, ACC\_STRICT标记

e实例的初始化方法只能ACC\_PUBLIC, ACC\_PRIVATE，ACC\_PROTECTED三选一，以及ACC\_VARARGS, ACC\_STRICT, ACC\_SYNTHETIC可选，不能有其他标志

f实例或接口的方法被隐式调用，只能设置ACC\_STATIC 和 ACC\_STRICT标志

2.5.3 <init>方法

2.5.4 <clinit>方法

2.6属性结构(attribute\_info)

2.6.1基本结构

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| attribute\_name\_index | u2 | 指向常量池一个CONSTANT\_Utf8\_info值的索引 |
| attribute\_length | u4 | 属性长度，长度不包括自身和attribute\_name\_index的长度 |
| info[attribute\_length] | u1 |  |

2.6.2属性类型

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 位置 | 首次版本 | 说明 |
| ConstantValue | field\_info | 45.3 |  |
| Code | method\_info | 45.3 |  |
| Exceptions | method\_info | 45.3 |  |
| SourceFile | classFile | 45.3 |  |
| LineNumberTable | code | 45.3 |  |
| LocalVariableTable | code | 45.3 |  |
| InnerClasses | classFile | 45.3 |  |
| Synthetic | classFile  field\_info  method\_info | 45.3 |  |
| Deprecated | classFile  field\_info  method\_info | 45.3 |  |
| EnclosingMethod | classFile | 49 |  |
| Signature | classFile  field\_info  method\_info | 49 |  |
| SourceDebugExtension | classFile | 49 |  |
| LocalVariableTypeTable | code | 49 |  |
| RuntimeVisibleAnnotations | classFile  field\_info  method\_info | 49 |  |
| RuntimeInvisibleAnnotations | classFile  field\_info  method\_info | 49 |  |
| RuntimeVisibleParameterAnnotations | method\_info | 49 |  |
| RuntimeInvisibleParameterAnnotations | method\_info | 49 |  |
| AnnotationDefault | method\_info | 49 |  |
| StackMapTable | code | 50 |  |
| BootstrapMethods | classFile | 51 |  |
| RuntimeVisibleTypeAnnotations | classFile  field\_info  method\_info  code | 52 |  |
| RuntimeInvisibleTypeAnnotations | classFile  field\_info  method\_info  code | 52 |  |
| MethodParameters | method\_info | 52 |  |
| Module | classFile | 53 |  |
| ModulePackages | classFile | 53 |  |
| ModuleMainClass | classFile | 53 |  |
| NestHost | classFile | 55 |  |
| NestMembers | classFile | 55 |  |

2.6.3属性类型结构

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 字段 | 字段类型 | 说明 |
| ConstantValue | field\_info | 45.3 |  |
| Code | method\_info | 45.3 |  |
| Exceptions | method\_info | 45.3 |  |
| SourceFile | classFile | 45.3 |  |
| LineNumberTable | code | 45.3 |  |
| LocalVariableTable | code | 45.3 |  |
| InnerClasses | classFile | 45.3 |  |
| Synthetic | classFile  field\_info  method\_info | 45.3 |  |
| Deprecated | classFile  field\_info  method\_info | 45.3 |  |
| EnclosingMethod | classFile | 49 |  |
| Signature | classFile  field\_info  method\_info | 49 |  |
| SourceDebugExtension | classFile | 49 |  |
| LocalVariableTypeTable | code | 49 |  |
| RuntimeVisibleAnnotations | classFile  field\_info  method\_info | 49 |  |
| RuntimeInvisibleAnnotations | classFile  field\_info  method\_info | 49 |  |
| RuntimeVisibleParameterAnnotations | method\_info | 49 |  |
| RuntimeInvisibleParameterAnnotations | method\_info | 49 |  |
| AnnotationDefault | method\_info | 49 |  |
| StackMapTable | code | 50 |  |
| BootstrapMethods | classFile | 51 |  |
| RuntimeVisibleTypeAnnotations | classFile  field\_info  method\_info  code | 52 |  |
| RuntimeInvisibleTypeAnnotations | classFile  field\_info  method\_info  code | 52 |  |
| MethodParameters | method\_info | 52 |  |
| Module | classFile | 53 |  |
| ModulePackages | classFile | 53 |  |
| ModuleMainClass | classFile | 53 |  |
| NestHost | classFile | 55 |  |
| NestMembers | classFile | 55 |  |

## 3类的加载

### 3.1类的加载器

3.1.1概述

3.1.2 双亲委派

### 3.2类加载过程

A close up of a piece of paper

Description automatically generated

#### 3.2.1概述

a类加载必须按加载、验证、准备、初始化、卸载顺序按部就班的开始，但有可能会在一个阶段执行的过程中调用、激活另一个阶段

b解析在一些情况下可以在 初始化 阶段以后开始

c加载阶段和连接阶段部分工作交叉进行

d

e

f

#### 3.2.2加载(Loading)

a什么时候开始加载，JVM规范没有强制约束

b接口或类C的创建时由另外个类或接口D触发或D调用JSE类库中的某些方法触发，比如反射等

c如果C是数组，则因数组类没有外部的二进制表示，所以都是JVM创建，不通过类加载器加载(created by the Java Virtual Machine rather than by a class loader)

d无论加载器L直接创建C或授权其他加载器创建C，都称L是C的初始加载器

e JVM支持两种加载器：JVM提供的引导类加载器、用户自定义的类加载器

f在C(或它的父类)的初始加载器抛出ClassNotFoundException错误时，JVM试图验证或解析C，必须抛出NoClassDefFoundError(*If the Java Virtual Machine ever attempts to load a class C during verification or resolution (but not initialization), and the class loader that is used to initiate loading of C throws an instance of ClassNotFoundException, then the Java Virtual Machine must throw an instance of NoClassDefFoundError whose cause is the instance of ClassNotFoundException*)

g一个功能良好的加载器应当保证三个行为：

g.1如果名称相同，返回的class对象必定相同

g.2无论是L1加载，还是L1委托给L2加载，都必须返回相同Class对象

g.3用户自定义加载器在加载出错时，必须反应加载出错点

h加载数组类A创建过程遵循以下规则

i加载阶段结束后，class文件就会被以指定格式存储在方法区，并在类型数据存放好后，会在堆中生成对应的Class类对象，作为访问方法区中类型数据的外部接口

#### 3.2.3验证(Verification)

3.2.3.1概述

a因为JVM虚拟机规范和《深入理解JAVA虚拟机规范(第三版)》对这块介绍差异较大，这里以JVM规范为主要介绍内容

b如果类或接口不满足静态或结构的约束，必须在失败处抛出VerifyError异常

3.2.3.2格式验证

a检查项包括但不限于：

a.1是否魔数开头

a.2主、次版本是否符合当前JVM范围

a.3常量池中是否存在 不存在或类型错误的常量

a.4常量类型的数据结构是否正确

a.5所有属性的数据结构是否正确

a.6不能缺失内容，尾部不能有多余数据

a.7所有字段/方法引用，必须有有效的名称、类、描述符

b 格式检查不保证某个字段/方法/描述符是否真实存在，只保证格式正确

c

3.2.3.3JVM代码约束

a静态约束

b结构化约束

3.2.3.4 Class文件校验

a类型检查验证

b类型推导验证

3.2.3.5 JVM限制

#### 3.2.4准备(Preparation)

#### 3.2.5解析(Resolution)

3.2.5.1概述

3.2.5.2解析类和接口

3.2.5.3解析字段

3.2.5.4解析方法

3.2.5.5解析接口方法

#### 3.2.6初始化(Initialization)

a、初始化阶段，就是执行构造器<clinit>()方法的过程

b、只有发生以下行为时，类或接口才会初始化(A class or interface C may be initialized only as a result of:)

b.1执行指定指令：new、getstatic，putstatic、invokestatic

b.2在初次调用java.lang.invoke.MethodHandle实例时，方法句柄是2 (REF\_getStatic), 4 (REF\_putStatic), 6 (REF\_invokeStatic), or 8 (REF\_newInvokeSpecial)

b.3调用反射方法

b.4对某个类的子类进行初始化

b.5如果接口c声明了一个非抽象、非静态的方法，有类直接或间接引用此接口的类初始化(*If C is an interface that declares a non-abstract, non-static method, the initialization of a class that implements C directly or indirectly.)*

b.6被指定为jvm启动时初始类

c在类或接口初始化之前，当前类或接口必须已经通过验证、准备阶段。

d直到初始化阶段，JVM才真正开始执行类中编写的Java代码，将主导权交给app

e每个类或接口都有个唯一的初始化锁，锁的实现有JVM实现自定义

f JVM必须保证一个类的初始化线程安全。

g初始化过程

注：

1:执行初始化过程，线程的中断状态不受影响

2:抛出的异常E，不是Error或其子类，那就为E创建个ExceptionInInitializerError实例代替它，如果OOM问题，则用OOM错误代替E

3:利用ConstantValue属性初始化类中每个final static字段(*Then, initialize each final static field of C with the constant value in its ConstantValue attribute, in the order the fields appear in the ClassFile structure*.)

4:初始化方法<clinit>

A close up of a map

Description automatically generated

### 3.3模块化

# 四、JVM字节码执行