**Class文件结构**

在加类加载子系统之前，我们需要首先了解Class文件的内部结构。   
Class文件是一种平台无关的二进制字节码格式(ByteCode)，该字节码能够被JVM识别并解释执行或被JVM进一步编译成本地机器码(Native Code)后执行。Class文件以8位字节为单位，排列紧凑中间没有任何分隔符。   
Class文件结构。

ClassFile {

//魔数 固定位0xcafebabe

u4 magic;

//Class文件版本号和JDK版本

u2 minor\_version;

u2 major\_version;

//常量池

u2 constant\_pool\_count;

cp\_info contant\_pool[constant\_pool\_count – 1];

//访问标记

u2 access\_flags;

//类的继承关系

u2 this\_class;

u2 super\_class;

//实现的接口

u2 interfaces\_count;

u2 interfaces[interfaces\_count];

//字段表集合

u2 fields\_count;

field\_info fields[fields\_count];

//方法表集合

u2 methods\_count;

method\_info methods[methods\_count];

u2 attributes\_count;

attribute\_info attributes[attributes\_count];

}

**常量池**

常量池主要存放字面量(Literal)和符号引用(Symbolic Reference)。   
字面量如：文本字符串(如：String “abc”)、声明为final的常量值。   
符号引用如：

* 类和接口的权限定名（如: “java.lang.Object”）
* 字段的名称和描述符
* 方法的名称和描述符

Java编译和C/C++不同，没有链接Linking步骤，**编译后的Class文件中只是方法和字段的符号引用，而没有真正内存地址**。当虚拟机运行时需要从常量池获取对应符号引用，**在类加载的才将符号引用翻译成具体地址**；甚至在方法多态情况下在每次方法执行时才翻译为具体地址。

常量池项目类型主要有

CONSTANT\_Utf8\_info     方法、字段等都需要用该类型常量描述   
CONSTANT\_String\_info    字符串类型字面量   
CONSTANT\_FieldRef\_info    字段符号引用   
CONSTANT\_MethodRef\_info    类中方法符号引用   
CONSTANT\_Class\_info    类或接口引用符号(类全限定名)

例子如下：

package org.jvminternals;

public class SimpleClass {

public void sayHello() {

System.out.println("Hello");

}

}

**对应常量池：**

public class org.jvminternals.SimpleClass

SourceFile: "SimpleClass.java"

minor version: 0

major version: 51

flags: ACC\_PUBLIC, ACC\_SUPER

Constant pool:

#1 = Methodref #6.#17 // java/lang/Object."<init>":()V

#2 = Fieldref #18.#19 // java/lang/System.out:Ljava/io/PrintStream;

#3 = String #20 // "Hello"

#4 = Methodref #21.#22 // java/io/PrintStream.println:(Ljava/lang/String;)V

#5 = Class #23 // org/jvminternals/SimpleClass

#6 = Class #24 // java/lang/Object

#7 = Utf8 <init>

#8 = Utf8 ()V

#9 = Utf8 Code

#10 = Utf8 LineNumberTable

#11 = Utf8 LocalVariableTable

#12 = Utf8 this

#13 = Utf8 Lorg/jvminternals/SimpleClass;

#14 = Utf8 sayHello

#15 = Utf8 SourceFile

#16 = Utf8 SimpleClass.java

#17 = NameAndType #7:#8 // "<init>":()V

#18 = Class #25 // java/lang/System

#19 = NameAndType #26:#27 // out:Ljava/io/PrintStream;

#20 = Utf8 Hello

#21 = Class #28 // java/io/PrintStream

#22 = NameAndType #29:#30 // println:(Ljava/lang/String;)V

#23 = Utf8 org/jvminternals/SimpleClass

#24 = Utf8 java/lang/Object

#25 = Utf8 java/lang/System

#26 = Utf8 out

#27 = Utf8 Ljava/io/PrintStream;

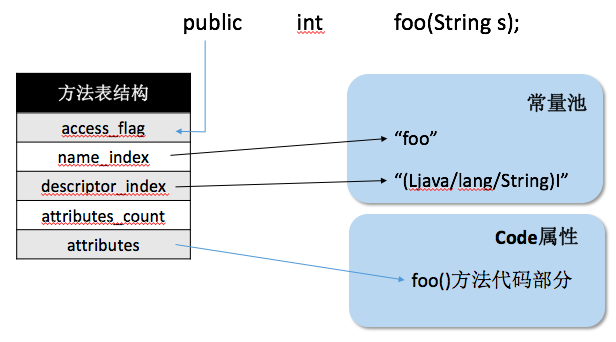
#28 = Utf8 java/io/PrintStream

#29 = Utf8 println

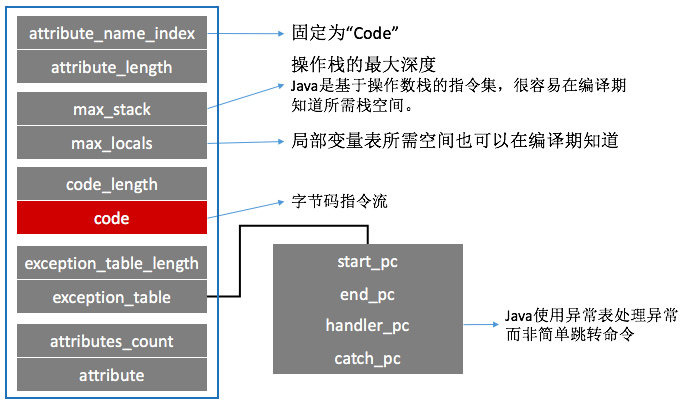
#30 = Utf8 (Ljava/lang/String;)V

**访问标志**

标示访问信息：是类还是接口？是否为public类型？是否定义abstract？   
  
**类索引、父类索引、实现的接口索引**   
**this\_class**: 在例子中是常量池偏移5，最终指向本类权限定名常量：org/jvminternals/SimpleClass   
**supr\_class**:父类，同理。本例：java/lang/Object   
**interfaces[interfaces\_count]**：接口同理，只不过Java支持实现多接口。   
  
**字段表集合**：类变量或者实例变量。

**方法表集合**   


**属性表集合**   
属性表集合中最重要的部分就是Code属性，用于java代码编译后的字节码指令。   
Code属性如下：



上面例子，编译后的字节码：

{

public org.jvminternals.SimpleClass();

Signature: ()V

flags: ACC\_PUBLIC

Code:

stack=1, locals=1, args\_size=1

0: aload\_0

1: invokespecial #1 // Method java/lang/Object."<init>":()V

4: return

public void sayHello();

Signature: ()V

flags: ACC\_PUBLIC

Code:

stack=2, locals=1, args\_size=1

0: getstatic #2 // Field java/lang/System.out:Ljava/io/PrintStream;

3: ldc #3 // String "Hello"

5: invokevirtual #4 // Method java/io/PrintStream.println:(Ljava/lang/String;)V

8: return

}