Atitit.java 虚拟机的构成 与指令分类 与 指令集合 以及字节码查看工具javjap

[1](#_Toc11536)*[.1.](#_Toc11536)* [虚拟机的构成](#_Toc11536) *[java虚拟机--处理器、堆栈、寄存器、指令系统。](#_Toc11536)*[1](#_Toc11536)

[1.2. 虚拟机执行过程 1](#_Toc32543)

[1.3. 约有250个指令 2](#_Toc24048)

[2. JVM指令助记符 分类 2](#_Toc12278)

[2.1. 变量到操作数栈： 2](#_Toc26148)

[2](#_Toc6088)*[.2. 算数指令](#_Toc6088)* [3](#_Toc6088)

[2](#_Toc15241)*[.3. 移位指令](#_Toc15241)* [3](#_Toc15241)

[2.4. 逻辑指令 4](#_Toc18094)

[2.5. 流程跳转指令 4](#_Toc8373)

[2.6. Oo指令 4](#_Toc17447)

[2.7. 运算指令 5](#_Toc6508)

[3. 查看指令反编译工具 6](#_Toc7742)

[3.1.   分析java语言特性的一个好帮手是使用javap工具查看java编译后的字节码， 6](#_Toc30613)

[3.2. 配置eclipse 的run外部tool 7](#_Toc4893)

## 虚拟机的构成 *****java虚拟机--处理器、堆栈、寄存器、指令系统。*****

 Java堆栈有堆栈块（stack frames (or frames)）组成。堆栈块包含Java方法调用的状态。当一个线程调用一个方法时，Java虚拟机会将一个新的块压到Java堆栈中，当这个方法运行结束时，Java虚拟机会将对应的块弹出并抛弃。   
     Java虚拟机不使用寄存器保存计算的中间结果，而是用Java堆栈在存放中间结果。这是的Java虚拟机的指令更紧凑，也更容易在一个没有寄存器的设备上实现Java虚拟机。

## 虚拟机执行过程

    Java虚拟机指令由一个字节长度的、代表某种特定含义的操作码(Opcode)以及其后的零个至多个代表此操作参数的操作数构成。虚拟机中许多指令并不包含操作数，只有一个操作码。若忽略异常，JVM解释器使用一下为代码即可有效工作。

复制代码 代码如下:

do{  
    自动计算PC寄存器以及从PC寄存器的位置取出操作码  
    if(存在操作数) 取出操作数;  
    执行操作码所定义的操作;  
}while(处理下一次循环)

操作数的数量以及长度，取决于操作码，若一个操作数长度超过了一个字节，将会以Big-Endian顺序存储(高位在前字节码)，其值应为(byte1<<8)|byte2。

作者:: 绰号:老哇的爪子 （ 全名：：Attilax Akbar Al Rapanui 阿提拉克斯 阿克巴 阿尔 拉帕努伊 ） 汉字名：艾龙，  EMAIL:1466519819@qq.com

转载请注明来源： http://blog.csdn.net/attilax

## 约有250个指令

参考

JVM指令集（指令码、助记符、功能描述） --- 逐渐更新 --- 2011-08-23 - 专栏 - 博客频道 - CSDN.NET.htm

# JVM指令助记符 分类

## 变量到操作数栈：

iload,iload\_,lload,lload\_,fload,fload\_,dload,dload\_,aload,aload\_  
操作数栈到变量：istore,istore\_,lstore,lstore\_,fstore,fstore\_,dstore,dstor\_,astore,astore\_  
常数到操作数栈：bipush,sipush,ldc,ldc\_w,ldc2\_w,aconst\_null,iconst\_ml,iconst\_*,lconst\_,fconst\_,dconst\_*

*把数据装载到操作数栈：baload,caload,saload,iaload,laload,faload,daload,aaload  
从操作数栈存存储到数组：bastore,castore,sastore,iastore,lastore,fastore,dastore,aastore*

## *算数指令*

*加：iadd,ladd,fadd,dadd  
减：is ,ls ,fs ,ds   
乘：imul,lmul,fmul,dmul  
除：idiv,ldiv,fdiv,ddiv  
余数：irem,lrem,frem,drem  
取负：ineg,lneg,fneg,dneg*

## *移位指令*

*移位：ishl,lshr,iushr,lshl,lshr,lushr  
按位或：ior,lor  
按位与：iand,land  
按位异或：ixor,lxor  
类型转换：i2l,i2f,i2d,l2f,l2d,f2d(放宽数值转换)  
          i2b,i2c,i2s,l2i,f2i,f2l,d2i,d2l,d2f(缩窄数值转换)*

## 逻辑指令

## 流程跳转指令

*有条件转移：ifeq,iflt,ifle,ifne,ifgt,ifge,ifnull,ifnonnull,if\_icmpeq,if\_icmpene,  
            if\_icmplt,if\_icmpgt,if\_icmple,if\_icmpge,if\_acmpeq,if\_acmpne,lcmp,fcmpl  
    fcmpg,dcmpl,dcmpg  
复合条件转移：tableswitch,lookupswitch  
无条件转移：goto,goto\_w,jsr,jsr\_w,ret  
方法返回：ireturn,lreturn,freturn,dreturn,areturn,return  
异常：athrow  
finally关键字的实现使用：jsr,jsr\_w,ret*

## Oo指令

*创建类实便：new  
创建新数组：newarray,anewarray,multianwarray  
访问类的域和类实例域：getfield,putfield,getstatic,putstatic  
获取数组长度：arraylength  
检相类实例或数组属性：instanceof,checkcast  
操作数栈管理：pop,pop2,dup,dup2,dup\_xl,dup2\_xl,dup\_x2,dup2\_x2,swap  
调度对象的实便方法：invokevirt l  
调用由接口实现的方法：invokeinterface  
调用需要特殊处理的实例方法：invokespecial  
调用命名类中的静态方法：invokestatic*  3)访问字段指令:getfield,putfield,getstatic,putstatic

  7)检查实例类型指令:instanceof,checkcast

## 运算指令

    算术指令用于对两个操作数栈上的值进行某种特定运算，并把结果重新存入到操作栈顶。  
    1)加法指令:iadd,ladd,fadd,dadd  
    2)减法指令:isub,lsub,fsub,dsub  
    3)乘法指令:imul,lmul,fmul,dmul  
    4)除法指令:idiv,ldiv,fdiv,ddiv  
    5)求余指令:irem,lrem,frem,drem  
    6)取反指令:ineg,leng,fneg,dneg  
    7)位移指令:ishl,ishr,iushr,lshl,lshr,lushr  
    8)按位或指令:ior,lor  
    9)按位与指令:iand,land  
    10)按位异或指令:ixor,lxor  
    11)局部变量自增指令:iinc  
    12)比较指令:dcmpg,dcmpl,fcmpg,fcmpl,lcmp

    Java虚拟机没有明确规定整型数据溢出的情况，但规定了

# 查看指令反编译工具

## 分析java语言特性的一个好帮手是使用javap工具查看java编译后的字节码，

楼主今天在学习java泛型中的桥方法时遇到一些不解，想到javap这个好工具可以帮助解答一些疑惑，索性就捣鼓如何在eclipse中配置javap工具快速查看java字节码。

选项：

-help 帮助

-l 输出行和变量的表

-public 只输出public方法和域

-protected 只输出public和protected类和成员

-package 只输出包，public和protected类和成员，这是默认的

-p -private 输出所有类和成员

-s 输出内部类型签名

-c 输出分解后的代码，例如，类中每一个方法内，包含java[字节码](http://baike.baidu.com/view/560330.htm" \t "http://baike.baidu.com/_blank)的指令，

-verbose 输出栈大小，方法参数的个数

-constants 输出静态final常量

## 配置eclipse 的run外部tool

Worikdir：：： ${project\_loc}

Argue：：：-classpath WEB-INF\classes -c ${java\_type\_name}

-classpath WEB-INF\classes -l -p -s -constants -v -c ${java\_type\_name}

**package** aaa;

**public** **class** haloDecomp {

**public** **static** **void** main(String[] args) {

System.***out***.println("helo world");

}

}

返回zai console：：

Compiled from "haloDecomp.java"

public class aaa.haloDecomp {

public aaa.haloDecomp();

Code:

0: aload\_0

1: invokespecial #8 // Method java/lang/Object."<init>":()V

4: return

public static void main(java.lang.String[]);

Code:

0: getstatic #16 // Field java/lang/System.out:Ljava/io/PrintStream;

3: ldc #22 // String helo world

5: invokevirtual #24 // Method java/io/PrintStream.println:(Ljava/lang/String;)V

8: return

}

Classfile /C:/workspace 空格/AtiPlatf/WEB-INF/classes/aaa/haloDecomp.class

Last modified 2016-2-8; size 755 bytes

MD5 checksum 6cd22bffd3cc27d213122a02856eb541

Compiled from "haloDecomp.java"

public class aaa.haloDecomp

SourceFile: "haloDecomp.java"

minor version: 0

major version: 50

flags: ACC\_PUBLIC, ACC\_SUPER

Constant pool:

#1 = Class #2 // aaa/haloDecomp

#2 = Utf8 aaa/haloDecomp

#3 = Class #4 // java/lang/Object

#4 = Utf8 java/lang/Object

#5 = Utf8 <init>

#6 = Utf8 ()V

#7 = Utf8 Code

#8 = Methodref #3.#9 // java/lang/Object."<init>":()V

#9 = NameAndType #5:#6 // "<init>":()V

#10 = Utf8 LineNumberTable

#11 = Utf8 LocalVariableTable

#12 = Utf8 this

#13 = Utf8 Laaa/haloDecomp;

#14 = Utf8 main

#15 = Utf8 ([Ljava/lang/String;)V

#16 = Fieldref #17.#19 // java/lang/System.out:Ljava/io/PrintStream;

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

#18 = Utf8 java/lang/System

#19 = NameAndType #20:#21 // out:Ljava/io/PrintStream;

#20 = Utf8 out

#21 = Utf8 Ljava/io/PrintStream;

#22 = String #23 // helo world

#23 = Utf8 helo world

#24 = Methodref #25.#27 // java/io/PrintStream.println:(Ljava/lang/String;)V

#25 = Class #26 // java/io/PrintStream

#26 = Utf8 java/io/PrintStream

#27 = NameAndType #28:#29 // println:(Ljava/lang/String;)V

#28 = Utf8 println

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

#30 = Methodref #1.#31 // aaa/haloDecomp.t2:()V

#31 = NameAndType #32:#6 // t2:()V

#32 = Utf8 t2

#33 = Utf8 args

#34 = Utf8 [Ljava/lang/String;

#35 = Class #36 // com/attilax/lang/AString

#36 = Utf8 com/attilax/lang/AString

#37 = Methodref #35.#38 // com/attilax/lang/AString."<init>":(Ljava/lang/String;)V

#38 = NameAndType #5:#29 // "<init>":(Ljava/lang/String;)V

#39 = Methodref #35.#40 // com/attilax/lang/AString.Left:(I)Lcom/attilax/lang/AString;

#40 = NameAndType #41:#42 // Left:(I)Lcom/attilax/lang/AString;

#41 = Utf8 Left

#42 = Utf8 (I)Lcom/attilax/lang/AString;

#43 = Methodref #25.#44 // java/io/PrintStream.println:(Ljava/lang/Object;)V

#44 = NameAndType #28:#45 // println:(Ljava/lang/Object;)V

#45 = Utf8 (Ljava/lang/Object;)V

#46 = Utf8 SourceFile

#47 = Utf8 haloDecomp.java

{

public aaa.haloDecomp();

Signature: ()V

flags: ACC\_PUBLIC

LineNumberTable:

line 5: 0

LocalVariableTable:

Start Length Slot Name Signature

0 5 0 this Laaa/haloDecomp;

Code:

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

0: aload\_0

1: invokespecial #8 // Method java/lang/Object."<init>":()V

4: return

LineNumberTable:

line 5: 0

LocalVariableTable:

Start Length Slot Name Signature

0 5 0 this Laaa/haloDecomp;

public static void main(java.lang.String[]);

Signature: ([Ljava/lang/String;)V

flags: ACC\_PUBLIC, ACC\_STATIC

LineNumberTable:

line 8: 0

line 9: 8

line 10: 11

LocalVariableTable:

Start Length Slot Name Signature

0 12 0 args [Ljava/lang/String;

Code:

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

0: getstatic #16 // Field java/lang/System.out:Ljava/io/PrintStream;

3: ldc #22 // String helo world

5: invokevirtual #24 // Method java/io/PrintStream.println:(Ljava/lang/String;)V

8: invokestatic #30 // Method t2:()V

11: return

LineNumberTable:

line 8: 0

line 9: 8

line 10: 11

LocalVariableTable:

Start Length Slot Name Signature

0 12 0 args [Ljava/lang/String;

public static void t2();

Signature: ()V

flags: ACC\_PUBLIC, ACC\_STATIC

LineNumberTable:

line 14: 0

line 15: 19

LocalVariableTable:

Start Length Slot Name Signature

Code:

stack=4, locals=0, args\_size=0

0: getstatic #16 // Field java/lang/System.out:Ljava/io/PrintStream;

3: new #35 // class com/attilax/lang/AString

6: dup

7: ldc #22 // String helo world

9: invokespecial #37 // Method com/attilax/lang/AString."<init>":(Ljava/lang/String;)V

12: iconst\_2

13: invokevirtual #39 // Method com/attilax/lang/AString.Left:(I)Lcom/attilax/lang/AString;

16: invokevirtual #43 // Method java/io/PrintStream.println:(Ljava/lang/Object;)V

19: return

LineNumberTable:

line 14: 0

line 15: 19

LocalVariableTable:

Start Length Slot Name Signature

}

参考

【JAVA】虚拟机指令集 - 老马睡不醒的日志 - 网易博客.htm

Java虚拟机工作原理 - - ITeye技术网站.htm

Java字节码指令集的使用详细\_java\_脚本之家.htm

javap（反汇编命令）详解 - Robin Hu的专栏 - 博客频道 - CSDN.NET.htm

JVM指令集（指令码、助记符、功能描述） --- 逐渐更新 --- 2011-08-23 - 专栏 - 博客频道 - CSDN.NET.htm

JVM指令集（指令码、助记符、功能描述） --- 逐渐更新 --- 2011-08-23 - 专栏 - 博客频道 - CSDN.NET.htm