**什么是[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "_blank)虚拟机：**

Java虚拟机是一个想象中的机器,在实际的计算机上通过软件模拟来实现。Java虚拟机有自己想象中的硬件,如处理器、堆栈、寄存器等,还具有相应的指令系统。Java虚拟机由五个部分组成:一组[指令集](http://baike.baidu.com/view/9174.htm" \t "_blank)、一组寄存器、一个[栈](http://baike.baidu.com/view/38877.htm)、一个无用单元收集堆(Garbage-collected-heap)、一个方法区域。这五部分是Java虚拟机的逻辑成份,不依赖任何实现技术或组织方式,但它们的功能必须在真实机器上以某种方式实现。

**1.Java指令集**　　Java虚拟机支持大约248个字节码。每个字节码执行一种基本的CPU运算,例如,把一个整数加到寄存器,子程序转移等。Java指令集相当于Java程序的汇编语言。

**2.寄存器**   
　　Java虚拟机的寄存器用于保存机器的运行状态,与微处理器中的某些专用寄存器类似。   
　　Java虚拟机的寄存器有四种:   
　　pc:Java程序计数器。   
　　optop:指向操作数栈顶端的指针。   
　　frame:指向当前执行方法的执行环境的指针。   
　　vars:指向当前执行方法的局部变量区第一个变量的指针。

**3.栈**

       java虚拟机的栈有三个区域:局部变量区、运行环境区、操作数区。

　　(1)局部变量区 每个Java方法使用一个固定大小的局部变量集。它们按照与vars寄存器的字偏移量来寻址。局部变量都是32位的。长整数和双精度浮点数占据了两个局部变量的空间,却按照第一个局部变量的索引来寻址。(例如,一个具有索引n的局部变量,如果是一个双精度浮点数,那么它实际占据了索引n和n+1所代表的存储空间。)虚拟机规范并不要求在局部变量中的64位的值是64位对齐的。虚拟机提供了把局部变量中的值装载到操作数栈的指令,也提供了把操作数栈中的值写入局部变量的指令。

　　(2)运行环境区 在运行环境中包含的信息用于动态链接,正常的方法返回以及异常传播。

**·动态链接**

　　运行环境包括对指向当前类和当前方法的解释器符号表的指针,用于支持方法代码的动态链接。方法的class文件代码在引用要调用的方法和要访问的变量时使用符号。动态链接把符号形式的方法调用翻译成实际方法调用,装载必要的类以解释还没有定义的符号,并把变量访问翻译成与这些变量运行时的存储结构相应的偏移地址。动态链接方法和变量使得方法中使用的其它类的变化不会影响到本程序的代码。

**·正常的方法返回**

　　如果当前方法正常地结束了,在执行了一条具有正确类型的返回指令时,调用的方法会得到一个返回值。执行环境在正常返回的情况下用于恢复调用者的寄存器,并把调用者的程序计数器增加一个恰当的数值,以跳过已执行过的方法调用指令,然后在调用者的执行环境中继续执行下去。

**·异常和错误传播**

　　异常情况在Java中被称作Error(错误)或Exception(异常),是Throwable类的子类,在程序中的原因是:①动态链接错,如无法找到所需的class文件。②运行时错,如对一个空指针的引用

**·程序使用了throw语句。**

　　当异常发生时,Java虚拟机采取如下措施:

　　·检查与当前方法相联系的catch子句表。每个catch子句包含其有效指令范围,能够处理的异常类型,以及处理异常的代码块地址。

　　·与异常相匹配的catch子句应该符合下面的条件:造成异常的指令在其指令范围之内,发生的异常类型是其能处理的异常类型的子类型。如果找到了匹配的catch子句,那么系统转移到指定的异常处理块处执行;如果没有找到异常处理块,重复寻找匹配的catch子句的过程,直到当前方法的所有嵌套的catch子句都被检查过。

　　·由于虚拟机从第一个匹配的catch子句处继续执行,所以catch子句表中的顺序是很重要的。因为Java代码是结构化的,因此总可以把某个方法的所有的异常处理器都按序排列到一个表中,对任意可能的程序计数器的值,都可以用线性的顺序找到合适的异常处理块,以处理在该程序计数器值下发生的异常情况。

　　·如果找不到匹配的catch子句,那么当前方法得到一个"未截获异常"的结果并返回到当前方法的调用者,好像异常刚刚在其调用者中发生一样。如果在调用者中仍然没有找到相应的异常处理块,那么这种错误传播将被继续下去。如果错误被传播到最顶层,那么系统将调用一个缺省的异常处理块。

　　(3)操作数栈区 机器指令只从操作数栈中取操作数,对它们进行操作,并把结果返回到栈中。选择栈结构的原因是:在只有少量寄存器或非通用寄存器的机器(如Intel486)上,也能够高效地模拟虚拟机的行为。操作数栈是32位的。它用于给方法传递参数,并从方法接收结果,也用于支持操作的参数,并保存操作的结果。例如,iadd指令将两个整数相加。相加的两个整数应该是操作数栈顶的两个字。这两个字是由先前的指令压进堆栈的。这两个整数将从堆栈弹出、相加,并把结果压回到操作数栈中。

　　每个原始数据类型都有专门的指令对它们进行必须的操作。每个操作数在栈中需要一个存储位置,除了long和double型,它们需要两个位置。操作数只能被适用于其类型的操作符所操作。例如,压入两个int类型的数,如果把它们当作是一个long类型的数则是非法的。在Sun的虚拟机实现中,这个限制由字节码验证器强制实行。但是,有少数操作(操作符dupe和swap),用于对运行时数据区进行操作时是不考虑类型的。

**4.无用单元收集堆**

　　Java的堆是一个运行时数据区,类的实例(对象)从中分配空间。Java语言具有无用单元收集能力:它不给程序员显式释放对象的能力。Java不规定具体使用的无用单元收集**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**,可以根据系统的需求使用各种各样的算法。

　　5.**方法区**

　　方法区与传统语言中的编译后代码或是Unix进程中的正文段类似。它保存方法代码(编译后的java代码)和符号表。在当前的Java实现中,方法代码不包括在无用单元收集堆中,但计划在将来的版本中实现。每个类文件包含了一个Java类或一个Java界面的编译后的代码。可以说类文件是Java语言的执行代码文件。为了保证类文件的平台无关性,Java虚拟机规范中对类文件的格式也作了详细的说明。其具体细节请参考Sun公司的Java虚拟机规范。

java编译执行过程：

机器码

Java虚拟机

解释器

Java 源代码

Java编译器

字节码

通俗说明：Java中引入了虚拟机的概念，即在机器和编译程序之间加入了一层抽象的虚拟的机器。这台虚拟的机器在任何平台上都提供给编译程序一个的共同的接口。编译程序只需要面向虚拟机，生成虚拟机能够理解的代码，然后由解释器来将虚拟机代码转换为特定系统的机器码执行。在Java中，这种供虚拟机理解的代码叫做字节码（ByteCode），它不面向任何特定的处理器，只面向虚拟机。每一种平台的解释器是不同的，但是实现的虚拟机是相同的。Java源程序经过编译器编译后变成字节码，字节码由虚拟机解释执行，虚拟机将每一条要执行的字节码送给解释器，解释器将其翻译成特定机器上的机器码，然后在特定的机器上运行。