链滴

从JVM编译优化角度解读String的赋值比较

作者：wanglei0622

原文链接：https://ld246.com/article/1461723989636

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

<h2>JVM工作流程</h2>
先JAVAC编译Java源文件为class字节码文件，class本身也描述不同java对象的格式的一种对象。java的class文件是统一的，是基于字节码的，它就是描述程序运行的虚指令的集合，而这个虚指令的集合与任何平台无关，JVM认识他，在运行时环境（JRE），jm里将他翻译成对应平台OS指令。进程启动时，通常会加载一些JVM核心库，并不会加载项目中所有LASS，程序访问类之前，需要先用jvm的classloader预加载，同一个classloader中一个类原则上只被加载一次，在常见的web容器中（如Tomcat）加载不同deploy（部署工程）会相互隔离，使用不的classloader来加载不同的deploy，也就是并非真正意义上的隔离，跨classloader访问还是可以做的.
<h2>String对象赋值比较的测试代码
</h2>
<pre class="brush: java">public class StringTest {
    static String a = "a";
    static String b = "b";
    static final String c = "a";
    static String d = "a"+"b";
    static String e = a+"b";
    static String f = c+"b" ;
    static String g = getA()+"b";
    static String h = new String(d);
    public static String getA(){
        return "a";
    }
    /**
     * jvm编译时优化原理,jvm有常量池，常量计算在编译时就已经获得结果，如"a"+"b"编译后字节中只有“ab”常量，运行时直接赋值,方法中的返回值不能确定，new String(“a”)新建对象也是运行时决定，也当作变量，final修饰的对象，
     * 引用只能被赋值一次，如果编译时不能确定赋值的内容，也当作变量，变量会在运行时新建对赋值。
     * java的基本类型比较，因为编译器的优化原理，编译时，节省常量池空间，编译时能确定的相常量只用一个引用地址，所以基本类型和其他常量的比较用==比较引用地址。
     * intern()是在常量池中用equals()遍历查找值和比较对象相等的引用地址返回，没有就创建一个值字符串。所以一定会返回相同的引用
     * String的equals()方法是重写Object的，在object中是直接用“==”的比较的，String的源码，先用==比较地址引用是否相等，在比较字符串长度，然后逐个字符（charA[i]==charB[i]比较。
     * String赋值 用“+”不一定比StringBuffer.append()慢，因为常量相加在编译时计算好结果了，而append()需要在运行时计算。
     */
    public static void main(String[] args) {
        System.out.println(d==e);//false
        System.out.println(d==f);//true
        System.out.println(d==g);//false
        System.out.println(d==h);//false
        System.out.println(d==g.intern());//true
    }

}

</pre>

== 在java中是引用地址的比较，java编译器在编译时会将包括基本数在内的常量，放在常量池中，所有常量对象引用相同的地址，常量的计算也会在编译时完成。方法中返回值如new String(“ab”)，或者return " a " ；编译时没有做优化，因为方法体化情况复杂，可能需要递归遍历才知道返回的是什么，即使返回的是常量，也是对常量的引用实现一

拷贝返回的，这份拷贝并不是final的。
<h2>i=i++的原理和String赋值时对象的创建个数</spa
></h2>
<pre class="brush: java">public class SelfAdd {

/**
 * i++是先操作，再自加，++i是先自加再操作
 * 但是c=c++，在JVM中是做了类似（int temp = c;c++; c=temp）的操作，temp不是真实存在的
量，是栈（后进先出）顶的数据拷贝
 * 最终得出c的值为1；
 */
public static void main(String[] args) {
    int a=1,b=1,c=1,d=1;
    a++;
    ++b;
    c=c++;//先赋值，再自加，应该是为2了，最终却还是等于一，自加没有成功
    d=++d;
    System.out.println(a+","+b+","+c+","+d);//结果2,2,1,2

    String a1="a"+"b";//创建一个对象
    String a2=new String("多大的");//创建两个个对象
    String str1 = "aaa"; //创建一个对象
    String str2 = "bbb";
    String str3 = "aaabbb";
    String str4 = "aaa" + "bbb";//不会产生新的字符串对象
    System.out.println(str3 == str4);//true
    str4 = str1 + "bbb";//会产生新的字符串对象
    System.out.println(str3 == str4);//false
    str4 = str1 + str2;//会产生新的字符串对象
    System.out.println(str3 == str4);//false
}

}</pre>

