javap是JDK自带的反汇编器，可以查看java编译器为我们生成的字节码。通过它，我们可以对照源代码和字节码，从而了解很多编译器内部的工作。  
语法：  
　　javap [ 命令选项 ] class. . .  
　　javap 命令用于解析类文件。其输出取决于所用的选项。若没有使用选项，javap 将输出传递给它的类的 public 域及方法。javap 将其输出到标准输出设备上。  
命令选项  
　　-help 输出 javap 的帮助信息。  
　　-l 输出行及局部变量表。  
　　-b 确保与 JDK 1.1 javap 的向后兼容性。  
　　-public 只显示 public 类及成员。  
　　-protected 只显示 protected 和 public 类及成员。  
　　-package 只显示包、protected 和 public 类及成员。这是缺省设置。  
　　-private 显示所有类和成员。  
　　-J[flag] 直接将 flag 传给运行时系统。  
　　-s 输出内部类型签名。  
　　-c 输出类中各方法的未解析的代码，即构成 Java 字节码的指令。  
　　-verbose 输出堆栈大小、各方法的 locals 及 args 数,以及class文件的编译版本  
　　-classpath[路径] 指定 javap 用来查找类的路径。如果设置了该选项，则它将覆盖缺省值或 CLASSPATH 环境变量。目录用冒号分隔。  
 　 -bootclasspath[路径] 指定加载自举类所用的路径。缺省情况下，自举类是实现核心 Java 平台的类，位于 jrelib下面。

　　-extdirs[dirs] 覆盖搜索安装方式扩展的位置。扩展的缺省位置是 jrelibext。

英文说明：

C:\>javap -help  
Usage: javap <options> <classes>...  
  
where options include:  
   -c                        Disassemble the code  
   -classpath <pathlist>     Specify where to find user class files  
   -extdirs <dirs>           Override location of installed extensions  
   -help                     Print this usage message  
   -J<flag>                  Pass <flag> directly to the runtime system  
   -l                        Print line number and local variable tables  
   -public                   Show only public classes and members  
   -protected                Show protected/public classes and members  
   -package                  Show package/protected/public classes  
                             and members (default)  
   -private                  Show all classes and members  
   -s                        Print internal type signatures  
   -bootclasspath <pathlist> Override location of class files loaded  
                             by the bootstrap class loader  
   -verbose                  Print stack size, number of locals and args for methods  
                             If verifying, print reasons for failure

示例：

下面也经典的StringBuilder代替String做字符串的例子。

**[java]** [view plain](http://blog.csdn.net/hantiannan/article/details/7659904) [copy](http://blog.csdn.net/hantiannan/article/details/7659904)

[print?](http://blog.csdn.net/hantiannan/article/details/7659904)

1. **public** **class** JAVAPTest {
2. **public** **static** **void** main(String[] args) {
4. }
6. **public** **static** String contactWithStringNoLoopNoPara() {
7. String s = "This is " + " my " + "first JAVAP test code.";
8. **return** s;
9. }
11. **public** **static** String contactWithStringNoLoop(**int** count) {
12. String s = "This is " + " my " + count + "th JAVAP test code.";
13. **return** s;
14. }
16. **public** **static** String contactWithStringLoop(**int** count) {
17. String s = "";
18. **for** (**int** i = 0; i < count; i++) {
19. s += i;
20. }
21. **return** s;
22. }
24. **public** **static** String contactWithStringBufferLoop(**int** count) {
25. StringBuffer sb = **new** StringBuffer();
26. **for** (**int** i = 0; i < count; i++) {
27. sb.append(i);
28. }
29. **return** sb.toString();
30. }
31. }

先编译：javac JAVAPTest.[**Java**](http://lib.csdn.net/base/java)

执行反编译：javap -c JAVAPTest         //注意这个地方不需要class后缀。

结果如下：

**[java]** [view plain](http://blog.csdn.net/hantiannan/article/details/7659904) [copy](http://blog.csdn.net/hantiannan/article/details/7659904)

[print?](http://blog.csdn.net/hantiannan/article/details/7659904)

1. Compiled from "JAVAPTest.java"
2. **public** **class** JAVAPTest **extends** java.lang.Object{
3. **public** JAVAPTest();
4. Code:
5. 0:   aload\_0
6. 1:   invokespecial   #1; //Method java/lang/Object."<init>":()V
7. 4:   **return**
9. **public** **static** **void** main(java.lang.String[]);
10. Code:
11. 0:   **return**
13. **public** **static** java.lang.String contactWithStringNoLoopNoPara();
14. Code:
15. 0:   ldc     #2; //String This is  my first JAVAP test code.
16. 2:   astore\_0
17. 3:   aload\_0
18. 4:   areturn
20. **public** **static** java.lang.String contactWithStringNoLoop(**int**);
21. Code:
22. 0:   **new**     #3; //class java/lang/StringBuilder
23. 3:   dup
24. 4:   invokespecial   #4; //Method java/lang/StringBuilder."<init>":()V
25. 7:   ldc     #5; //String This is  my
26. 9:   invokevirtual   #6; //Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
27. 12:  iload\_0
28. 13:  invokevirtual   #7; //Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
29. 16:  ldc     #8; //String th JAVAP test code.
30. 18:  invokevirtual   #6; //Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
31. 21:  invokevirtual   #9; //Method java/lang/StringBuilder.toString:()Ljava/lang/String;
32. 24:  astore\_1
33. 25:  aload\_1
34. 26:  areturn
36. **public** **static** java.lang.String contactWithStringLoop(**int**);
37. Code:
38. 0:   ldc     #10; //String
39. 2:   astore\_1
40. 3:   iconst\_0
41. 4:   istore\_2
42. 5:   iload\_2
43. 6:   iload\_0
44. 7:   if\_icmpge       35
45. 10:  **new**     #3; //class java/lang/StringBuilder
46. 13:  dup
47. 14:  invokespecial   #4; //Method java/lang/StringBuilder."<init>":()V
48. 17:  aload\_1
49. 18:  invokevirtual   #6; //Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
50. 21:  iload\_2
51. 22:  invokevirtual   #7; //Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
52. 25:  invokevirtual   #9; //Method java/lang/StringBuilder.toString:()Ljava/lang/String;
53. 28:  astore\_1
54. 29:  iinc    2, 1
55. 32:  **goto**    5
56. 35:  aload\_1
57. 36:  areturn
59. **public** **static** java.lang.String contactWithStringBufferLoop(**int**);
60. Code:
61. 0:   **new**     #11; //class java/lang/StringBuffer
62. 3:   dup
63. 4:   invokespecial   #12; //Method java/lang/StringBuffer."<init>":()V
64. 7:   astore\_1
65. 8:   iconst\_0
66. 9:   istore\_2
67. 10:  iload\_2
68. 11:  iload\_0
69. 12:  if\_icmpge       27
70. 15:  aload\_1
71. 16:  iload\_2
72. 17:  invokevirtual   #13; //Method java/lang/StringBuffer.append:(I)Ljava/lang/StringBuffer;
73. 20:  pop
74. 21:  iinc    2, 1
75. 24:  **goto**    10
76. 27:  aload\_1
77. 28:  invokevirtual   #14; //Method java/lang/StringBuffer.toString:()Ljava/lang/String;
78. 31:  areturn
80. }

有这个结果我们可以知道。

1。contactWithStringNoLoopNoPara方法中，代码里面是字符串拼接，貌似需要是用StringBuilder替换的好。其实在看了上面的反编译结果后，已经自动组合成一个固定字符串了。因此完全没有必要使用StringBuilder。

**[java]** [view plain](http://blog.csdn.net/hantiannan/article/details/7659904) [copy](http://blog.csdn.net/hantiannan/article/details/7659904)

[print?](http://blog.csdn.net/hantiannan/article/details/7659904)

1. 0:   ldc     #2; //String This is  my first JAVAP test code.

2。contactWithStringNoLoop方法中，因为使用到了变量，貌似需要是用StringBuilder替换的好。其实在看了上面的反编译结果后，已经自动使用了StringBuilder。所以代码也没有必要使用StringBuilder。

3. contactWithStringLoop方法中，是循环拼接字符串，貌似需要是用StringBuilder替换的好。看了反编译后，每个循环里面都各自生成了一个StringBuilder，并将StringBuilder.toString()防赋值给我们的Sring变量。而我们希望的是只生成一个StringBuilder对象。因此改为StringBuilder的好。循环的时候改为contactWithBufferLoop的方法最好。

4.contactWithBufferLoop方法中，是循环拼接字符串。也是我们预想的步骤在执行。