Usando javap na classe java

A melhor maneira de desempenho de concatenação de string é utilizar a classe StringBuffer.
Entretanto, as maiorias dos desenvolvedores nunca viram a diferença nas instruções de bytecode.
Foi incluído um programa (executável) dentro do Java Development Kit (JDK) chamado javap, para mostrar para você.
Javap lê a classe java e mostra as informações sobre seus métodos.
Ele não descompila um programa fonte em java, só le as instruções de bytecode definida pela especificação da máquina virtual Java.
Javap é útil quando você quer ver o que seu compilador está fazendo, ou quando você quiser ver, que efeito uma mudança no código, terá na linha compilada da classe.
Vamos usar a classe StringBuffer contra a classe String.
Abaixo está uma classe que contém dois métodos que retornam uma string, que tem como parâmetro um int, onde n é fornecido quem acessa o método.
A única diferença entre os dois métodos é que um usa a classe String, e o outro usa a classe StringBuffer.

```
public class Javap {

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

  private static String mString(int n) {
    String s = "";
    for(int i = 0; i < n; i++) {
      s += i;
    }
    return s;
  }

  private static String mStringBuffer(int n) {
    StringBuffer sb = new StringBuffer();
    for(int i = 0; i < n; i++) {
      sb.append(i);
    }
    return sb.toString();
  }
}
```

Vamos olhar agora a saída do javap com a opção -c que diz ao javap para mostrar o bytecode encontrado na classe.
Execute a seguinte instrução:

```
C:\>javap -c Javap
```

A saída do comando é mostrada a seguir:

```
Method java.lang.String mString(int)
   0 ldc #2 <String "">
   2 astore_1
   3 iconst_0
   4 istore_2
   5 goto 30
   8 new #3 <Class java.lang.StringBuffer>
  11 dup
  12 invokespecial #4 <Method java.lang.StringBuffer()>
  15 aload_1
  16 invokevirtual #5 <Method java.lang.StringBuffer append(java.lang.String)>
  19 iload_2
  20 invokevirtual #6 <Method java.lang.StringBuffer append(int)>
  23 invokevirtual #7 <Method java.lang.String toString()>
  26 astore_1
  27 iinc 2 1
  30 iload_2
  31 iload_0
  32 if_icmplt 8
  35 aload_1
  36 areturn

Method java.lang.String mStringBuffer(int)
   0 new #3 <Class java.lang.StringBuffer>
   3 dup
   4 invokespecial #4 <Method java.lang.StringBuffer()>
   7 astore_1
   8 iconst_0
   9 istore_2
  10 goto 22
  13 aload_1
  14 iload_2
  15 invokevirtual #6 <Method java.lang.StringBuffer append(int)>
  18 pop
  19 iinc 2 1
  22 iload_2
  23 iload_0
  24 if_icmplt 13
  27 aload_1
  28 invokevirtual #7 <Method java.lang.String toString()>
  31 areturn
```

A saída é um pouco cripitada, se você nunca viu o assembler do Java, você pode ver que o método mString cria uma nova instância de StringBuffer cada vez através do laço.
O n, adiciona o valor atual da string existente ao StringBuffer e adiciona o valor atual do laço. Finalmente, chama toString e cria uma nova referência a string existente.
Isto está ao contrário do método mStringBuffer, que chama somente o StringBuffer existente, chamando o método append cada vez através do laço.

Não há nenhuma criação do objeto e nenhuma referência nova a string.
Neste caso, nós sabemos que usar a classe StringBuffer em vez da classe String é uma boa idéia. Se não souber, o javap pode nos ajudar encontrar a resposta.
Você não encontrará frequentemente nas circunstâncias para querer um descompilador Java. Mas você tem já um em sua máquina e que é simples usar.
Se você tiver interessado, de uma olhada nas outras opções do javap.
Você poderá encontrar as opções, que lhe agrada em seu ambiente.

Atualmente presta serviço de consultoria Java para Unimed de Santos.

Luís Carlos Moreira da Costa
Consultor Técnico Java/C++
tcljava@ig.com.br
http://www.tclsoftware.com.br