理解 String 及 String.intern() 在实际中的应用

文章地址

博客分类：

- J2EE

ApacheTomcatJVMperformance

Java 代码

```
1. 1. 首先 String 不属于 8 种基本数据类型，String 是一个对象。
2.
3.     因为对象的默认值是 null，所以 String 的默认值也是 null；但它又是一种特殊的对象，有其它对象没有的一些特性。
4.
5.     2. new String()和 new String("")都是申明一个新的空字符串，是空串不是 null；
6.
7.     3. String str="kvill";
8. String str=new String ("kvill");的区别：
9.
10.     在这里，我们不谈堆，也不谈栈，只先简单引入常量池这个简单的概念。
11.
12.     常量池(constant pool)指的是在编译期被确定，并被保存在已编译的.class 文件中的一些数据。它包括了关于类、方法、接口等中的常量，也包括字符串常量。
13.
14.     看例 1：
15.
16. String s0="kvill";
17. String s1="kvill";
18. String s2="kv" + "ill";
19. System.out.println( s0==s1 );
20. System.out.println( s0==s2 );
21.
22.     结果为：
23.
24. true
25. true
26.
27.     首先，我们要知道 Java 会确保一个字符串常量只有一个拷贝。
28.
29.     因为例子中的 s0 和 s1 中的"kvill"都是字符串常量，它们在编译期就被确定了，所以 s0==s1 为 true；而"kv"和"ill"也都是字符串常量，当一个字符串由多个字符串常量连接而成时，它自己肯定也是字符串常量，所以 s2 也同样在编译期就被解析为一个字符串常量，所以 s2 也是常量池中"kvill"的一个引用。
30.
31.     所以我们得出 s0==s1==s2;
```

32.

33. 用 new String() 创建的字符串不是常量，不能在编译期就确定，所以 new String() 创建的字符串不放入常量池中，它们有自己的地址空间。

34.

35. 看例 2：

36.

```
37. String s0=”kvill”;
38. String s1=new String(”kvill”);
39. String s2=”kv” + new String(“ill”);
40. System.out.println( s0==s1 );
41. System.out.println( s0==s2 );
42. System.out.println( s1==s2 );
```

43.

44. 结果为：

45.

```
46. false
47. false
48. false
```

49.

50. 例 2 中 s0 还是常量池中”kvill”的应用，s1 因为无法在编译期确定，所以是运行时创建的新对象”kvill”的引用，s2 因为有后半部分 new String(“ill”)所以也无法在编译期确定，所以也是一个新创建对象”kvill”的应用;明白了这些也就知道为何得出此结果了。

51.

52. 4. String.intern()：

53.

54. 再补充介绍一点：存在于.class 文件中的常量池，在运行期被 JVM 装载，并且可以扩充。String 的 intern()方法就是扩充常量池的一个方法；当一个 String 实例 str 调用 intern()方法时，Java 查找常量池中是否有相同 Unicode 的字符串常量，如果有，则返回其的引用，如果没有，则在常量池中增加一个 Unicode 等于 str 的字符串并返回它的引用；看例 3 就清楚了

55.

56. 例 3：

57.

```
58. String s0= “kvill”;
59. String s1=new String(”kvill”);
60. String s2=new String(“kvill”);
61. System.out.println( s0==s1 );
62. System.out.println( “**********” );
63. s1.intern();
64. s2=s2.intern(); //把常量池中“kvill”的引用赋给 s2
65. System.out.println( s0==s1);
66. System.out.println( s0==s1.intern() );
67. System.out.println( s0==s2 );
```

68.

结果为：

```
false
**********
false //虽然执行了 s1.intern(),但它的返回值没有赋给 s1
true //说明 s1.intern()返回的是常量池中”kvill”的引用
true
```

最后我再破除一个错误的理解：

有人说，“使用 String.intern()方法则可以将一个 String 类的保存到一个全局 String 表中，如果具有相同值的 Unicode 字符串已经在这个表中，那么该方法返回表中已有字符串的地址，如果在表中没有相同值的字符串，则将自己的地址注册到表中“如果我把他说的这个全局的 String 表理解为常量池的话，他的最后一句话，“如果在表中没有相同值的字符串，则将自己的地址注册到表中”是错的：

看例 4：

```
String s1=new String("kvill");
String s2=s1.intern();
System.out.println( s1==s1.intern() );
System.out.println( s1+" "+s2 );
System.out.println( s2==s1.intern() );
```

结果：

```
false
kvill kvill
true
```

在这个类中我们没有声名一个”kvill”常量，所以常量池中一开始是没有”kvill”的，当我们调用 s1.intern()后就在常量池中新添加了一个”kvill”常量，原来的不在常量池中的”kvill”仍然存在，也就不是“将自己的地址注册到常量池中”了。

s1==s1.intern()为 false 说明原来的“kvill”仍然存在；

s2 现在为常量池中“kvill”的地址，所以有 s2==s1.intern()为 true。

5. 关于 equals()和==:

这个对于 String 简单来说就是比较两字符串的 Unicode 序列是否相当，如果相等返回 true;而==是比较两字符串的地址是否相同，也就是是否是同一个字符串的引用。

6. 关于 String 是不可变的

```
106.
107.     这一说又要说很多，大家只要知道 String 的实例一旦生成就不会再改变了，比如说：Str
    ing str=”kv”+”ill”+” “+”ans”;
108. 就是有 4 个字符串常量，首先”kv”和”ill”生成了”kvill”存在内存中，然后”kvill”又
    和” “ 生成 ”kvill “存在内存中，最后又和生成了”kvill ans”;并把这个字符串的地址赋给
    了 str,就是因为 String 的“不可变”产生了很多临时变量，这也就是为什么建议用 StringBuff
    er 的原因了，因为 StringBuffer 是可改变的
```

出处：http://www.iteye.com/topic/122206

By the way,关于 String.intern() 在实际中的应用，我在 tomcat 的源码中找到了一个地方用到了，如下：

Java 代码

```
1.  /*
2.   * Copyright 1999,2004-2005 The Apache Software Foundation.
3.   *
4.   * Licensed under the Apache License, Version 2.0 (the "License");
5.   * you may not use this file except in compliance with the License.
6.   * You may obtain a copy of the License at
7.   *
8.   *      http://www.apache.org/licenses/LICENSE-2.0
9.   *
10.  * Unless required by applicable law or agreed to in writing, software
11.  * distributed under the License is distributed on an "AS IS" BASIS,
12.  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13.  * See the License for the specific language governing permissions and
14.  * limitations under the License.
15.  * ====================================================================
16.  *
17.  * This software consists of voluntary contributions made by many
18.  * individuals on behalf of the Apache Software Foundation and was
19.  * originally based on software copyright (c) 1999, International
20.  * Business Machines, Inc., http://www.apache.org.  For more
21.  * information on the Apache Software Foundation, please see
22.  * <http://www.apache.org/>.
23.  */
24.
25. package org.apache.jasper.xmlparser;
26.
27. /**
28.  * This class is a symbol table implementation that guarantees that
29.  * strings used as identifiers are unique references. Multiple calls
```

```
30.  * to <code>addSymbol</code> will always return the same string
31.  * reference.
32.  * <p>
33.  * The symbol table performs the same task as <code>String.intern()</code>
34.  * with the following differences:
35.  * <ul>
36.  *  <li>
37.  *   A new string object does not need to be created in order to
38.  *   retrieve a unique reference. Symbols can be added by using
39.  *   a series of characters in a character array.
40.  *  </li>
41.  *  <li>
42.  *   Users of the symbol table can provide their own symbol hashing
43.  *   implementation. For example, a simple string hashing algorithm
44.  *   may fail to produce a balanced set of hashcodes for symbols
45.  *   that are <em>mostly</em> unique. Strings with similar leading
46.  *   characters are especially prone to this poor hashing behavior.
47.  *  </li>
48.  * </ul>
49.  *
50.  * @author Andy Clark
51.  * @version $Id: SymbolTable.java 306179 2005-07-27 15:12:04Z yoavs $
52.  */
53. public class SymbolTable {
54.
55.     //
56.     // Constants
57.     //
58.
59.     /** Default table size. */
60.     protected static final int TABLE_SIZE = 101;
61.
62.     //
63.     // Data
64.     //
65.
66.     /** Buckets. */
67.     protected Entry[] fBuckets = null;
68.
69.     // actual table size
70.     protected int fTableSize;
71.
72.     //
73.     // Constructors
```

```
74.     //
75.
76.     /** Constructs a symbol table with a default number of buckets. */
77.     public SymbolTable() {
78.         this(TABLE_SIZE);
79.     }
80.
81.     /** Constructs a symbol table with a specified number of buckets. */
82.     public SymbolTable(int tableSize) {
83.         fTableSize = tableSize;
84.         fBuckets = new Entry[fTableSize];
85.     }
86.
87.     //
88.     // Public methods
89.     //
90.
91.     /**
92.      * Adds the specified symbol to the symbol table and returns a
93.      * reference to the unique symbol. If the symbol already exists,
94.      * the previous symbol reference is returned instead, in order
95.      * guarantee that symbol references remain unique.
96.      *
97.      * @param symbol The new symbol.
98.      */
99.     public String addSymbol(String symbol) {
100.
101.         // search for identical symbol
102.         int bucket = hash(symbol) % fTableSize;
103.         int length = symbol.length();
104.         OUTER: for (Entry entry = fBuckets[bucket]; entry != null; entr
    y = entry.next) {
105.             if (length == entry.characters.length) {
106.                 for (int i = 0; i < length; i++) {
107.                     if (symbol.charAt(i) != entry.characters[i]) {
108.                         continue OUTER;
109.                     }
110.                 }
111.                 return entry.symbol;
112.             }
113.         }
114.
115.         // create new entry
116.         Entry entry = new Entry(symbol, fBuckets[bucket]);
```

```
117.          fBuckets[bucket] = entry;
118.          return entry.symbol;
119.
120.      } // addSymbol(String):String
121.
122.      /**
123.       * Adds the specified symbol to the symbol table and returns a
124.       * reference to the unique symbol. If the symbol already exists,
125.       * the previous symbol reference is returned instead, in order
126.       * guarantee that symbol references remain unique.
127.       *
128.       * @param buffer The buffer containing the new symbol.
129.       * @param offset The offset into the buffer of the new symbol.
130.       * @param length The length of the new symbol in the buffer.
131.       */
132.      public String addSymbol(char[] buffer, int offset, int length) {
133.
134.          // search for identical symbol
135.          int bucket = hash(buffer, offset, length) % fTableSize;
136.          OUTER: for (Entry entry = fBuckets[bucket]; entry != null; entr
    y = entry.next) {
137.              if (length == entry.characters.length) {
138.                  for (int i = 0; i < length; i++) {
139.                      if (buffer[offset + i] != entry.characters[i]) {
140.                          continue OUTER;
141.                      }
142.                  }
143.                  return entry.symbol;
144.              }
145.          }
146.
147.          // add new entry
148.          Entry entry = new Entry(buffer, offset, length, fBuckets[bucke
    t]);
149.          fBuckets[bucket] = entry;
150.          return entry.symbol;
151.
152.      } // addSymbol(char[],int,int):String
153.
154.      /**
155.       * Returns a hashcode value for the specified symbol. The value
156.       * returned by this method must be identical to the value returned
157.       * by the <code>hash(char[],int,int)</code> method when called
158.       * with the character array that comprises the symbol string.
```

```
159.      *
160.      * @param symbol The symbol to hash.
161.      */
162.     public int hash(String symbol) {
163.
164.         int code = 0;
165.         int length = symbol.length();
166.         for (int i = 0; i < length; i++) {
167.             code = code * 37 + symbol.charAt(i);
168.         }
169.         return code & 0x7FFFFF;
170.
171.     } // hash(String):int
172.
173.     /**
174.      * Returns a hashcode value for the specified symbol information.
175.      * The value returned by this method must be identical to the value
176.      * returned by the <code>hash(String)</code> method when called
177.      * with the string object created from the symbol information.
178.      *
179.      * @param buffer The character buffer containing the symbol.
180.      * @param offset The offset into the character buffer of the start
181.      *               of the symbol.
182.      * @param length The length of the symbol.
183.      */
184.     public int hash(char[] buffer, int offset, int length) {
185.
186.         int code = 0;
187.         for (int i = 0; i < length; i++) {
188.             code = code * 37 + buffer[offset + i];
189.         }
190.         return code & 0x7FFFFF;
191.
192.     } // hash(char[],int,int):int
193.
194.     /**
195.      * Returns true if the symbol table already contains the specified
196.      * symbol.
197.      *
198.      * @param symbol The symbol to look for.
199.      */
200.     public boolean containsSymbol(String symbol) {
201.
202.         // search for identical symbol
```

```
203.         int bucket = hash(symbol) % fTableSize;
204.         int length = symbol.length();
205.         OUTER: for (Entry entry = fBuckets[bucket]; entry != null; entr
    y = entry.next) {
206.             if (length == entry.characters.length) {
207.                 for (int i = 0; i < length; i++) {
208.                     if (symbol.charAt(i) != entry.characters[i]) {
209.                         continue OUTER;
210.                     }
211.                 }
212.                 return true;
213.             }
214.         }
215.
216.         return false;
217.
218.     } // containsSymbol(String):boolean
219.
220.     /**
221.      * Returns true if the symbol table already contains the specified
222.      * symbol.
223.      *
224.      * @param buffer The buffer containing the symbol to look for.
225.      * @param offset The offset into the buffer.
226.      * @param length The length of the symbol in the buffer.
227.      */
228.     public boolean containsSymbol(char[] buffer, int offset, int lengt
    h) {
229.
230.         // search for identical symbol
231.         int bucket = hash(buffer, offset, length) % fTableSize;
232.         OUTER: for (Entry entry = fBuckets[bucket]; entry != null; entr
    y = entry.next) {
233.             if (length == entry.characters.length) {
234.                 for (int i = 0; i < length; i++) {
235.                     if (buffer[offset + i] != entry.characters[i]) {
236.                         continue OUTER;
237.                     }
238.                 }
239.                 return true;
240.             }
241.         }
242.
243.         return false;
```

```
244.
245.     } // containsSymbol(char[],int,int):boolean
246.
247.     //
248.     // Classes
249.     //
250.
251.     /**
252.      * This class is a symbol table entry. Each entry acts as a node
253.      * in a linked list.
254.      */
255.     protected static final class Entry {
256.
257.         //
258.         // Data
259.         //
260.
261.         /** Symbol. */
262.         public String symbol;
263.
264.         /**
265.          * Symbol characters. This information is duplicated here for
266.          * comparison performance.
267.          */
268.         public char[] characters;
269.
270.         /** The next entry. */
271.         public Entry next;
272.
273.         //
274.         // Constructors
275.         //
276.
277.         /**
278.          * Constructs a new entry from the specified symbol and next entry
279.          * reference.
280.          */
281.         public Entry(String symbol, Entry next) {
282.             this.symbol = symbol.intern();
283.             characters = new char[symbol.length()];
284.             symbol.getChars(0, characters.length, characters, 0);
285.             this.next = next;
286.         }
287.
```

```
288.         /**
289.          * Constructs a new entry from the specified symbol information an
    d
290.          * next entry reference.
291.          */
292.         public Entry(char[] ch, int offset, int length, Entry next) {
293.             characters = new char[length];
294.             System.arraycopy(ch, offset, characters, 0, length);
295.             symbol = new String(characters).intern();
296.             this.next = next;
297.         }
298. 
299.     } // class Entry
300. 
301. } // class SymbolTable
```


localhost == 127.0.0.1 ? | 动态加载 js

- 2009-12-16 11:33
- 浏览 6403
- 评论(1)
- 分类:编程语言
- 相关推荐

评论

1 楼 vk14311 2014-03-06

String s1=new String("kvill");
String s2=s1.intern();
System.out.println(s1==s1.intern());
System.out.println(s1+" "+s2);
System.out.println(s2==s1.intern());

这个例子应该改为：

String s1=new String("kv") + "ill";
String s2=s1.intern();
System.out.println(s1==s1.intern());
System.out.println(s1+" "+s2);
System.out.println(s2==s1.intern());

因为在 String s1=new String("kvill"); 这句的时候已经存在"kvill"常量了。