.NET 面试题解析(03)-string 与字符串操作

系列文章目录地址：

.NET 面试题解析(00)-开篇来谈谈面试 & 系列文章索引

字符串可以说是 C#开发中最常用的类型了，也是对系统性能影响很关键的类型，熟练掌握字符串的操作非常重要。

常见面试题目：

1.字符串是引用类型类型还是值类型？

2.在字符串连接处理中，最好采用什么方式，理由是什么？

3.使用 StringBuilder 时，需要注意些什么问题？

4.以下代码执行后内存中会存在多少个字符串？分别是什么？输出结果是什么？为什么呢？

```
string st1 = "123" + "abc";
string st2 = "123abc";
Console.WriteLine(st1 == st2);
Console.WriteLine(System.Object.ReferenceEquals(st1, st2));
```

5.以下代码执行后内存中会存在多少个字符串？分别是什么？输出结果是什么？为什么呢？

```
string s1 = "123";
string s2 = s1 + "abc";
string s3 = "123abc";
Console.WriteLine(s2 == s3);
Console.WriteLine(System.Object.ReferenceEquals(s2, s3));
```

6.使用 C#实现字符串反转算法，例如：输入"12345"，输出"54321"。

7.下面的代码输出结果？为什么？

```
object a = "123";
object b = "123";
Console.WriteLine(System.Object.Equals(a,b));
Console.WriteLine(System.Object.ReferenceEquals(a,b));
string sa = "123";
Console.WriteLine(System.Object.Equals(a, sa));
```

```
Console.WriteLine(System.Object.ReferenceEquals(a, sa));
```

深入浅出字符串操作

string 是一个特殊的引用类型，使用上有点像值类型。之所以特殊，也主要是因为 string 太常用了，为了提高性能及开发方便，对 string 做了特殊处理，给予了一些专用特性。为了弥补 string 在字符串连接操作上的一些性能不足，便有了 StringBuilder。

认识 string

首先需要明确的，string 是一个引用类型，其对象值存储在托管堆中。string 的内部是一个 char 集合，他的长度 Length 就是字符 char 数组的字符个数。string 不允许使用 new string()的方式创建实例，而是另一种更简单的语法，直接赋值（string aa= “000”这一点也类似值类型）。

认识 string，先从一个简单的示例代码入手：

```
public void DoStringTest()
{
    var aa = "000";
    SetStringValue(aa);
    Console.WriteLine(aa);
}

private void SetStringValue(string aa)
{
    aa += "111";
}
```

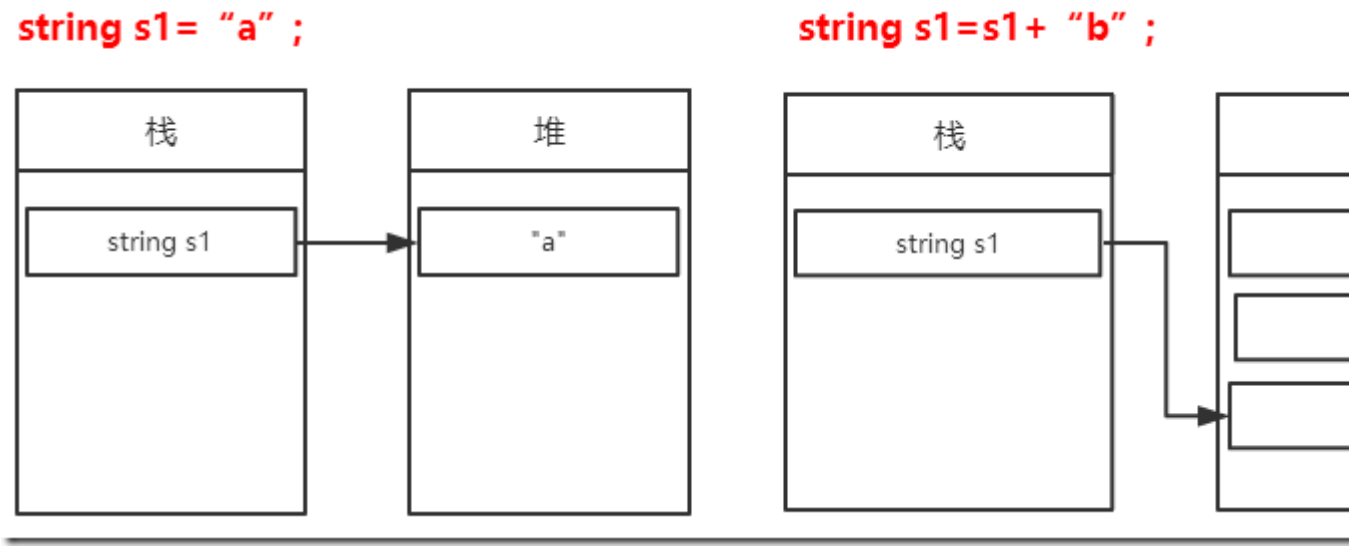
上面的输出结果为“000”。

通过前面的值类型与引用类型的文章，我们知道 string 是一个引用类型，既然是一个引用类型，参数传递的是引用地址，那为什么不是输出“000111”呢？是不是很有值类型的特点呢！这一切的原因源于 string 类型的两个重要的特性：**恒定性**与**驻留性**

String 的恒定性（不变性）

字符串是不可变的，字符串一经创建，就不会改变，任何改变都会产生新的字符串。比如下面的代码，堆上先创建了字符串 s1=”a”，加上一个字符串“b”后，堆上会存在三个个字符串实例，如下图所示。

```
string s1 = "a";
string s2 = s1 + "b";
```



上文中的”**任何改变都会产生新的字符串**“，包括字符串的一些操作函数，如 str1.ToLower，Trim()，Remove(int startIndex, int count)，ToUpper()等，都会产生新的字符串，因此在很多编程实践中，对于字符串忽略大小的比较：

```
if（str1.ToLower()==str2.ToLower()） //这种方式会产生新的字符串，不推荐
if（string. Compare(str1,str2,true)） //这种方式性能更好
```

String 的驻留性

由于字符串的不变性，在大量使用字符串操作时，会导致创建大量的字符串对象，带来极大的性能损失。因此 CLR 又给 string 提供另外一个法宝，就是字符串驻留，先看看下面的代码，字符串 s1、s2 竟然是同一个对象！

```
var s1 = "123";
var s2 = "123";
Console.WriteLine(System.Object.Equals(s1, s2));  //输出 True
Console.WriteLine(System.Object.ReferenceEquals(s1, s2));  //输出 True
```

相同的字符串在内存（堆）中只分配一次，第二次申请字符串时，发现已经有该字符串是，直接返回已有字符串的地址，这就是驻留的基本过程。

字符串驻留的基本原理：

- CLR 初始化时会在内存中创建一个驻留池，内部其实是一个哈希表，存储被驻留的字符串和其内存地址。
- 驻留池是进程级别的，多个 AppDomain 共享。同时她不受 GC 控制，生命周期随进程，意思就是不会被 GC 回收（不回收！难道不会造成内存爆炸吗？不要急，且看下文）

- 当分配字符串时，首先会到驻留池中查找，如找到，则返回已有相同字符串的地址，不会创建新字符串对象。如果没有找到，则创建新的字符串，并把字符串添加到驻留池中。

如果大量的字符串都驻留到内存里，而得不到释放，不是很容易造成内存爆炸吗，当然不会了？因为不是任何字符串都会驻留，只有通过 IL 指令 `ldstr` 创建的字符串才会留用。

字符串创建的有多种方式，如下面的代码：

```
var s1 = "123";
var s2 = s1 + "abc";
var s3 = string.Concat(s1, s2);
var s4 = 123.ToString();
var s5 = s2.ToUpper();
```

其 IL 代码如下

```
.method public hidebysig instance void  ValueTypeTest() cil managed
{
  .custom instance void [nunit.framework]NUnit.Framework.TestAttribute::.ctor() = (
  // 代码大小       48 (0x30)
  .maxstack  2
  .locals init ([0] string s1,
           [1] string s2,
           [2] string s3,
           [3] string s4,
           [4] string s5,
           [5] int32 CS$0$0000)
  IL_0000:  nop
  IL_0001:  ldstr      "123"
  IL_0006:  stloc.0
  IL_0007:  ldloc.0
  IL_0008:  ldstr      "abc"
  IL_000d:  call       string [mscorlib]System.String::Concat(string,
                                                              string)
  IL_0012:  stloc.1
  IL_0013:  ldloc.0
  IL_0014:  ldloc.1
  IL_0015:  call       string [mscorlib]System.String::Concat(string,
                                                              string)
  IL_001a:  stloc.2
  IL_001b:  ldc.i4.s   123
  IL_001d:  stloc.s    CS$0$0000
  IL_001f:  ldloca.s   CS$0$0000
  IL_0021:  call       instance string [mscorlib]System.Int32::ToString()
  IL_0026:  stloc.3
  IL_0027:  ldloc.1
  IL_0028:  callvirt   instance string [mscorlib]System.String::ToUpper()
  IL_002d:  stloc.s    s5
  IL_002f:  ret
} // end of method BlogTest::ValueTypeTest
```

在上面的代码中，出现两个字符串常量，“123”和“abc”，这个两个常量字符串在 IL 代码中都是通过 IL 指令 `ldstr` 创建的，只有该指令创建的字符串才会被驻留，其他方式产生新的字符串都不会被驻留，也就不会共享字符串了，会被 GC 正常回收。

那该如何来验证字符串是否驻留呢，string 类提供两个静态方法：

- String.Intern(string str) 可以主动驻留一个字符串；
- String.IsInterned(string str);检测指定字符串是否驻留，如果驻留则返回字符串，否则返回 NULL

```
public static string Intern(string str);
public static string IsInterned(string str);
```

请看下面的示例代码

```
var s1 = "123";
var s2 = s1 + "abc";
Console.WriteLine(s2);   //输出：123abc
Console.WriteLine(string.IsInterned(s2) ?? "NULL");   //输出：NULL。因为“123abc”没有驻留

string.Intern(s2);   //主动驻留字符串
Console.WriteLine(string.IsInterned(s2) ?? "NULL");   //输出：123abc
```

认识 StringBuilder

大量的编程实践和意见中，都说大量字符串连接操作，应该使用 StringBuilder。相对于 string 的不可变，StringBuilder 代表可变字符串，不会像字符串，在托管堆上频繁分配新对象，StringBuilder 是个好同志。

首先 StringBuilder 内部同 string 一样，有一个 **char[]字符数组**，负责维护字符串内容。因此，与 char 数组相关，就有两个很重要的属性：

- public int **Capacity**：StringBuilder 的容量，其实就是字符数组的长度。
- public int **Length**：StringBuilder 中实际字符的长度，>=0，<=容量 Capacity。

StringBuilder 之所以比 string 效率高，主要原因就是不会创建大量的新对象，StringBuilder 在以下两种情况下会分配新对象：

- **追加字符串时，当字符总长度超过了当前设置的容量 Capacity，这个时候，会重新创建一个更大的字符数组，此时会涉及到分配新对象。**
- **调用 StringBuilder.ToString()，创建新的字符串。**

追加字符串的过程：

- StringBuilder 的默认初始容量为 16；
- 使用 stringBuilder.Append()追加一个字符串时，当字符数大于 16，StringBuilder 会自动申请一个更大的字符数组，一般是倍增；
- 在新的字符数组分配完成后，将原字符数组中的字符复制到新字符数组中，原字符数组就被无情的抛弃了（会被 GC 回收）；
- 最后把需要追加的字符串追加到新字符数组中；

简单来说，当 StringBuilder 的容量 **Capacity** 发生变化时，就会引起托管对象申请、内存复制等操作，带来不好的性能影响，因此设置合适的初始容量是非常必要的，尽量减少内存申请和对象创建。代码简单来验证一下：

```
StringBuilder sb1 = new StringBuilder();
```

```
Console.WriteLine("Capacity={0}; Length={1};", sb1.Capacity, sb1.Length); //输出：Capacity=16; Length=0;   //初始容量为 16
sb1.Append('a', 12);    //追加 12 个字符
Console.WriteLine("Capacity={0}; Length={1};", sb1.Capacity, sb1.Length); //输出：Capacity=16; Length=12;
sb1.Append('a', 20);    //继续追加 20 个字符，容量倍增了
Console.WriteLine("Capacity={0}; Length={1};", sb1.Capacity, sb1.Length); //输出：Capacity=32; Length=32;
sb1.Append('a', 41);    //追加 41 个字符，新容量=32+41=73
Console.WriteLine("Capacity={0}; Length={1};", sb1.Capacity, sb1.Length); //输出：Capacity=73; Length=73;

StringBuilder sb2 = new StringBuilder(80); //设置一个合适的初始容量
Console.WriteLine("Capacity={0}; Length={1};", sb2.Capacity, sb2.Length); //输出：Capacity=80; Length=0;
sb2.Append('a', 12);
Console.WriteLine("Capacity={0}; Length={1};", sb2.Capacity, sb2.Length); //输出：Capacity=80; Length=12;
sb2.Append('a', 20);
Console.WriteLine("Capacity={0}; Length={1};", sb2.Capacity, sb2.Length); //输出：Capacity=80; Length=32;
sb2.Append('a', 41);
Console.WriteLine("Capacity={0}; Length={1};", sb2.Capacity, sb2.Length); //输出：Capacity=80; Length=73;
```

为什么少量字符串不推荐使用 StringBuilder 呢？因为 StringBuilder 本身是有一定的开销的，少量字符串就不推荐使用了，使用 String.Concat 和 String.Join 更合适。

高效的使用字符串

- 在使用线程锁的时候，不要锁定一个字符串对象，因为字符串的驻留性，可能会引发不可以预料的问题；
- 理解字符串的不变性，尽量避免产生额外字符串，如：

```
if（str1.ToLower()==str2.ToLower()） //这种方式会产生新的字符串，不推荐
if（string. Compare(str1,str2,true)） //这种方式性能更好
```

- 在处理大量字符串连接的时候，尽量使用 StringBuilder，在使用 StringBuilder 时，尽量设置一个合适的长度初始值；
- 少量字符串连接建议使用 String.Concat 和 String.Join 代替。

题目答案解析:

1.字符串是引用类型类型还是值类型？

引用类型。

2.在字符串连加处理中，最好采用什么方式，理由是什么？

少量字符串连接，使用 String.Concat，大量字符串使用 StringBuilder，因为 StringBuilder 的性能更好，如果 string 的话会创建大量字符串对象。

3.使用 StringBuilder 时，需要注意些什么问题？

- 少量字符串时，尽量不要用，StringBuilder 本身是有一定性能开销的；
- 大量字符串连接使用 StringBuilder 时，应该设置一个合适的容量；

4.以下代码执行后内存中会存在多少个字符串？分别是什么？输出结果是什么？为什么呢？

```
string st1 = "123" + "abc";
string st2 = "123abc";
Console.WriteLine(st1 == st2);
Console.WriteLine(System.Object.ReferenceEquals(st1, st2));
```

输出结果：

```
True
True
```

内存中的字符串只有一个“123abc”,第一行代码（string st1 = "123" + "abc"; ）常量字符串相加会被编译器优化。由于字符串驻留机制，两个变量 st1、st2 都指向同一个对象。IL 代码如下：

```
.maxstack  2
.locals init ([0] string st1,
         [1] string st2)
IL_0000:  nop
IL_0001:  ldstr      "123abc"
IL_0006:  stloc.0
IL_0007:  ldstr      "123abc"
IL_000c:  stloc.1
IL_000d:  ldloc.0
IL_000e:  ldloc.1
IL_000f:  call       bool [mscorlib]System.String::op_Equality(string,
                                                               string)
IL_0014:  call       void [mscorlib]System.Console::WriteLine(bool)
IL_0019:  nop
IL_001a:  ldloc.0
IL_001b:  ldloc.1
IL_001c:  call       bool [mscorlib]System.Object::ReferenceEquals(object,
                                                                   object)
IL_0021:  call       void [mscorlib]System.Console::WriteLine(bool)
IL_0026:  nop
IL_0027:  ret
```

5.以下代码执行后内存中会存在多少个字符串？分别是什么？输出结果是什么？为什么呢？

```
string s1 = "123";
string s2 = s1 + "abc";
string s3 = "123abc";
Console.WriteLine(s2 == s3);
Console.WriteLine(System.Object.ReferenceEquals(s2, s3));
```

和第 5 题的结果肯定是不一样的，答案留给读者吧，文章太长了，写的好累！

6.使用 C#实现字符串反转算法，例如：输入"12345"，输出"54321"

这是一道比较综合的考察字符串操作的题目，答案可以有很多种。通过不同的答题可以看出程序猿的基础水平。下面是网上比较认可的两种答案，效率上都是比较不错的。

```
public static string Reverse(string str)
{
    if (string.IsNullOrEmpty(str))
    {
        throw new ArgumentException("参数不合法");
    }
```

```
    StringBuilder sb = new StringBuilder(str.Length);  //注意：设置合适的初始长度，可以显著提高效率（避免了多次内存申请）
    for (int index = str.Length - 1; index >= 0; index--)
    {
        sb.Append(str[index]);
    }
    return sb.ToString();
}
public static string Reverse(string str)
{
    if (string.IsNullOrEmpty(str))
    {
        throw new ArgumentException("参数不合法");
    }
    char[] chars = str.ToCharArray();
    int begin = 0;
    int end = chars.Length - 1;
    char tempChar;
    while (begin < end)
    {
        tempChar = chars[begin];
        chars[begin] = chars[end];
        chars[end] = tempChar;
        begin++;
        end--;
    }
    string strResult = new string(chars);
    return strResult;
}
```

还有一个比较简单也挺有效的方法：

```
public static string Reverse(string str)
{
    char[] arr = str.ToCharArray();
    Array.Reverse(arr);
    return new string(arr);
}
```

7.下面的代码输出结果？为什么？

```
object a = "123";
```

```
object b = "123";
Console.WriteLine(System.Object.Equals(a,b));
Console.WriteLine(System.Object.ReferenceEquals(a,b));
string sa = "123";
Console.WriteLine(System.Object.Equals(a, sa));
Console.WriteLine(System.Object.ReferenceEquals(a, sa));
```

输出结果全是 True，因为他们都指向同一个字符串实例，使用 object 声明和 string 声明在这里并没有区别（string 是引用类型）。

使用 object 声明和 string 声明到底有没有区别呢？，有点疑惑，一个朋友在面试时面试官有问过这个问题，那个面试官说 sa、a 是有区别的，且不相等。对于此疑问，欢迎交流。

.NET 面试题解析(00)-开篇来谈谈面试 & 系列文章索引

参考资料：

书籍：CLR via C#

书籍：你必须知道的.NET

深入理解 string 和如何高效地使用 string： http://www.cnblogs.com/artech/archive/2007/05/06/737130.html

C#基础知识梳理系列九：StringBuilder：http://www.cnblogs.com/solan/archive/2012/08/06/CSharp09.html

分类: C#.NET

/*梦里花落知多少*/

关注 - 5

粉丝 - 136

+加关注

« 上一篇：.NET 面试题解析(02)-拆箱与装箱

» 下一篇：.NET 面试题解析(04)-类型、方法与继承

posted @ 2016-03-04 09:24 /*梦里花落知多少*/ 阅读(856) 评论(11) 编辑 收藏

评论列表

#1 楼 2016-03-04 09:33 小菜鸟_code

楼主理解的很透彻啊

支持(0)反对(0)

#2 楼 2016-03-04 10:41 雨之竹

String 的恒定性（不变性）

string s1 = "a";

string s2 = a + "b";

应该有笔误

支持(1)反对(0)

#3 楼 2016-03-04 13:55 丨渊丨

我擦， 博主 一天一更啊 太勤快了 顺便问一下，有没有 socket 的面试题解析呢

支持(0)反对(0)

#4 楼[楼主] 2016-03-04 22:40 /*梦里花落知多少*/

@雨之竹

的确，观察的好细致，

支持(0)反对(0)

#5 楼[楼主] 2016-03-04 22:40 /*梦里花落知多少*/

@丨渊丨

那个不是很熟

支持(0)反对(0)

#6 楼 2016-03-04 22:43 丨渊丨

@/*梦里花落知多少*/

。。。。。。。。。。。。。。。。。。。

支持(0)反对(0)

#7 楼 2016-03-06 11:12 大橙子小橘子

驻留和我在其他地方看到的字符串池是一个概念对吧，用 new 关键字显式的去分配字符串，就不会去采用这个机制，期待你的下一篇

支持(0)反对(0)

#8 楼[楼主] 2016-03-06 12:29 /*梦里花落知多少*/

@大橙子小橘子

对的

支持(0)反对(0)

#9 楼 2016-03-08 11:05 闯哥

楼主的文章我都仔细看了，帮我这个初学者理了一下知识，谢谢

支持(0)反对(0)

#10 楼 2016-03-08 16:41 风行影者

使用 String.Contac，是不是写错了，应该是 String.Concat 吧

支持(0)反对(0)

#11 楼[楼主] 2016-03-08 18:04 /*梦里花落知多少*/

@风行影者

3q，已修正