第8章 string型とchar型　その1

string型は、文字列を取り扱います。

すでに、暗黙のうちに文字列を取り扱ってきました。

文字列リテラルは、ダブルクォーテーションで囲みます。

文字列はすべてオブジェクト扱いとなります。文字列オブジェクトを作成するには、

```
string str = "abc";
```

のように、文字列リテラルを利用します。これにより、"abc"という文字列オブジェクトがメモリ上に確保され、その参照がstr変数に格納されます。

文字列型に対してchar型というのもあります。文字リテラルはシングルクォートで囲みます。

```
char mychar = 'a';
```

上のように宣言すると、char型の変数mycharが確保され、文字aのユニコードがmycharに直接格納されます。

従ってstring型は参照型、char型は値型であるといえます。また、C#のstring型は、.NET型のSystem.String型、char型はSystem.Char型です。もちろんusing System;とあったらSystemを省略できます。

char型の配列から、文字列を作成することもできます。char型配列の作り方は、他の型の配列の作り方と同じです。

```
char[] 配列名 = new char[要素数];
```

char型配列から文字列オブジェクトを作成するには、

```
string 変数名 = new string(char型配列名);
```

のようにします。

文字列型オブジェクトは、Lengthプロパティを利用できます。

これは、オブジェクトの文字数を取得できます。たとえば

```
stinrg str = "abc";
int n = str.Length;
```

とすると、nには、文字列"abc"の文字数3が格納されます。

また、文字列型オブジェクトを配列のようにして

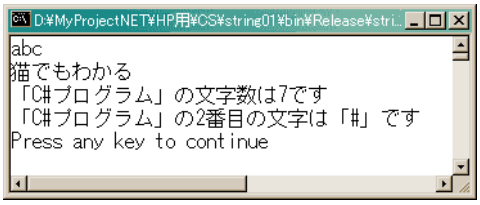
```
文字列型オブジェクト[n]
```

とすると、この文字列のn+1番目の文字を取得できます。あとの章で解説しますが「インデクサ」というものを使っています。

```
// string01.cs

using System;

class string01
{
    public static void Main()
    {
        char[] chararray = new char[3];
        chararray[0] = 'a';
        chararray[1] = 'b';
        chararray[2] = 'c';

        string str;
        str = new string(chararray);
        Console.WriteLine(str);

        char[] title = {'猫', 'で', 'も', 'わ', 'か', 'る'};
        string strTitle = new string(title);
        Console.WriteLine(strTitle);


        string strx = "C#プログラム";
        int n = strx.Length;
        Console.WriteLine("「{0}」の文字数は{1}です", strx, n);

        char c = strx[1];
        Console.WriteLine("「{0}」の2番目の文字は「{1}」です", strx, c);
```

```
    }
}
```

実行結果は次のようになります。



文字列型オブジェクトは、さらにCopyToメソッドを利用できます。

```
public void CopyTo(
   int sourceIndex,
   char[] destination,
   int destinationIndex,
   int count
);
```

これは、文字列型オブジェクトから任意の部分文字列をchar型の配列にコピーします。

sourceIndexは、このオブジェクトのコピー開始の位置(先頭文字が0)。

destinationは、コピー先のchar型配列。

destinationIndexは、コピー先の配列のどの位置からコピーするかを指定。

countは、コピーする文字数です。

ちょっと面倒そうに見えますが、使い方は簡単です。

```
// string02.cs

using System;

class string02
{
    public static void Main()
    {
        string str = "猫でもわかるプログラミング";
        char[] c = new char[7];

        //strの(6 + 1)番目の文字から、配列cへコピーします。
        //コピー先はインデックス0から7文字だけ受け入れます。
        str.CopyTo(6, c, 0, 7);
        Console.WriteLine(c);
    }
}
```

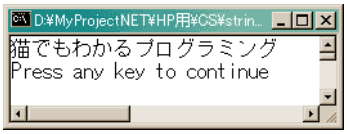
実行結果は次のようになります。

これを利用して、char型配列に文字列オブジェクトをまるごとコピーすることもできます。

```
// string03.cs

using System;

class string03
{
    public static void Main()
    {
        string str = "猫でもわかるプログラミング";
        char[] c = new char[str.Length];

        str.CopyTo(0, c, 0, str.Length);
        string str2 = new string(c);

        Console.WriteLine(str2);
```

```
    }
}
```

実行結果は次のようになります。



配列cにstrオブジェクトの内容を全部コピーしてから、この配列を利用してstr2オブジェクトを作成しています。

これと、似たものにSubstringメソッドがあります。これは、オブジェクトから部分文字列を取得します。これには、2つのバージョンがあります。

```
public string Substring(
   int startIndex
)
```

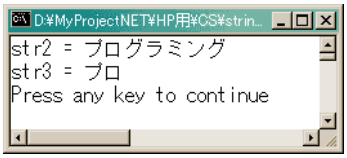
これは、指定のインデックスから最後までの文字列を取得します。

```
public string Substring(
   int startIndex,
   int length
)
```

これは、startIndexからlength文字だけの部分文字列を取得します。

```
// string04.cs

using System;

class string04
{
    public static void Main()
    {
        string str = "猫でもわかるプログラミング";
        string str2, str3;

        str2 = str.Substring(6);
        str3 = str.Substring(6, 2);

        Console.WriteLine("str2 = {0}", str2);
        Console.WriteLine("str3 = {0}", str3);
    }
}
```

実行結果は次のようになります。



str2はstrの(6 + 1)番目の文字から最後までです。

str3はstrの(6 + 1)番目の文字から2文字だけのコピーです。

文字列を丸ごとコピーするには、もっと簡単な方法があります。

stringクラスのCopy静的メソッドを使います。

```
public static string Copy(
   string str
);
```

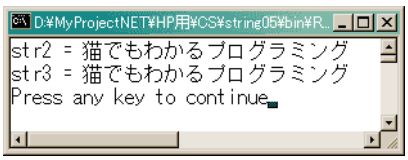
ここで、ちょっと注意しなくてはいけないことは、このメソッドは静的メソッドである点です。これは、オブジェクト名.メソッド名()で呼び出すのではなく、クラス名.メソッド名()で呼び出します。

Console.WriteLineと同じ呼び出し方法です。この件に関しては後の章で詳しく解説する予定なので、今は呼び出し方法に違いがあるということだけを覚えておいてください。

```
// string05.cs

using System;

class string05
{
    public static void Main()
    {
        string str = "猫でもわかるプログラミング";
        string str2, str3;

        str2 = string.Copy(str);
```

```
        Console.WriteLine("str2 = {0}", str2);

        str3 = str2;
        Console.WriteLine("str3 = {0}", str3);
    }
}
```

実行結果は、次のようになります。



str2はstring.Copyメソッドを利用してstrをコピーしています。

str3は、str2の参照を代入しています。結果としてはこれでもコピーと同じですね。

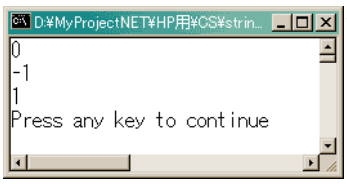
文字列を比較するには、CompareToメソッドを利用します。

```
public int CompareTo(
   string strB
);
```

このオブジェクトとstrBとを比較します。同じなら0が返されます。このオブジェクトがstrBより小さい場合負の値が、大きいなら正の値が返されます。

```
// string06.cs

using System;

class string06
{
    public static void Main()
    {
        string str1 = "abc", str2 = "abc", str3 = "bcd", str4 = "5";
        str2 = "abc";

        Console.WriteLine(str1.CompareTo(str2));
        Console.WriteLine(str1.CompareTo(str3));
        Console.WriteLine(str1.CompareTo(str4));
    }
}
```

実行結果は、次のようになります。



+/-は、辞書式に並べた時の位置と考えてもよいですね。

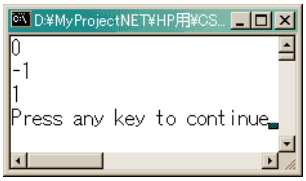
stringクラスには、CompareToメソッドと同じ働きをする静的メソッドもあります。

```
public static int Compare(
   string strA,
   string strB
);
```

strAとstrBが等しいときは0,strAが大きいときは正、小さいときは負の値を返します。

```
// string07.cs

using System;

class string07
{
    public static void Main()
    {
        string str1 = "abc", str2 = "abc", str3 = "bcd", str4 = "5";

        Console.WriteLine(string.Compare(str1, str2));
        Console.WriteLine(string.Compare(str1, str3));
        Console.WriteLine(string.Compare(str1, str4));
    }
}
```

実行結果は次のようになります。



結果は、string06.csの時と同じです。

[C# Index] [**総合**Index] [Previous Chapter] [Next Chapter]

Update 11/Aug/2006 By Y.Kumei