عنوان: **ایجاد رشته Alphanumeric تصادفی در سی شارپ**
نویسنده: امیر هاشم زاده
تاریخ: ۱۳۹۲/۰۲/۲۰ ۱۹:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, .NET, Random, alphanumeric

برای ایجاد یک رشته تصادفی Alphanumeric (شامل حرف و عدد) روشهای زیادی وجود دارد ولی در اینجا به تشریح 2 روش آن اکتفا می‌کنیم.

روش کلی: ابتدا بازه رشته تصادفی مورد نظر را تعیین می‌کنیم. سپس به اندازه طول رشته، اندیس تصادفی ایجاد می‌کنیم و بوسیله آنها کاراکتر تصادفی را از بازه بدست می‌آورم و در انتها کاراکترهای تصادفی را با هم ادغام کرده تا رشته نهایی حاصل شود.
روش اول:

ابتدا بازه (char) رشته را مشخص می‌کنیم.

```
var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
```

سپس بوسیله LINQ آن را به اندازه طول رشته دلخواه (در این مثال 8کاراکتر) تکرار می‌کنیم و برای انتخاب تصادفی یک کاراکتر در هر بازه (char) تکرار شده از کلاس جهت بدست آوردن اندیس تصادفی بازه استفاده می‌کنیم.

```
var random = new Random();
var result = new string(
Enumerable.Repeat(chars, 8)
          .Select(s => s[random.Next(s.Length)])
          .ToArray());
```

توجه: از این روش برای هیچ کدام از موارد مهم و کلیدی مانند ساخت کلمه عبور و توکن استفاده نکنید.
روش دوم:

همانند روش اول ابتدا بازه رشته را تعیین می‌کنیم.

```
char[] chars = new char[62];
chars="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToCharArray();
```

بعد از تعریف بازه، یک سری اعداد تصادفی غیر صفر را بوسیله کلاس RNGCryptoServiceProvider و متد GetNonZeroBytes آن، در متغیری که قرار است بعدا در ایجاد رشته‌ی تصادفی نیاز است پر می‌کنیم.

```
byte[] data = new byte[maxSize];
RNGCryptoServiceProvider crypto = new RNGCryptoServiceProvider();
crypto.GetNonZeroBytes(data);
```

در این مرحله به تعداد طول رشته تصادفی مورد نظر عدد تصادفی بین 0 تا 255 ذخیره شده در متغیر data داریم، برای ایجاد اندیس تصادفی از باقیمانده عدد تصادفی ایجاد شده در مرحله قبل (byte) به طول بازه (chars.Length) استفاده می‌کنیم سپس کاراکترهای تصادفی را کنار یکدیگر قرار می‌دهیم.

```
StringBuilder result = new StringBuilder(maxSize);
foreach (byte b in data)
{
  result.Append(chars[b % (chars.Length)]);
}
```

و در نهایت متد ما جهت ایجاد رشته Alphanumeric در روش دوم به شکل زیر خواهد بود:

```
public static string GetRandomAlphaNumeric (int maxSize)
{
      char[] chars = new char[62];
      chars ="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToCharArray();
      RNGCryptoServiceProvider crypto = new RNGCryptoServiceProvider();
      byte[] data = new byte[maxSize];
      crypto.GetNonZeroBytes(data);
      StringBuilder result = new StringBuilder(maxSize);
      foreach (byte b in data)
      {
           result.Append(chars[b % (chars.Length)]);
      }
      return result.ToString();
}
```

لازم به یادآوری است که رشته ایجاد شده در 2روش بیان شده **منحصر بفرد** نیست بلکه **تصادفی** است، درواقع تصادفی بودن با منحصر بودن متفاوت است، برای ایجاد رشته‌های منحصر بفرد روشهایی (البته این روشها 100 درصد نیستند ولی از قابلیت اطمینان بالایی برخوردار هستند) وجود دارد که پست‌های بعدی به آنها اشاره خواهم کرد.

عنوان: **اثبات قانون مشاهده‌گر در برنامه نویسی**
نویسنده: میثم نوایی
تاریخ: ۱۳۹۳/۰۷/۲۷ ۱:۳۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Threading, Random

امروز حین کدنویسی به یک مشکل نادر برخورد کردم. کلاسی پایه داشتم (مثلا Person) که یک سری کلاس دیگر از آن ارث بری میکردند (مثلا کلاس‌های Student و Teacher).در اینجا در کلاس پایه بصورت اتوماتیک یک ویژگی(Property) را روی کلاس‌های مشتق شده مقدار دهی میکردم؛ مثلا به این شکل:

```
public class Person
    {
        public Person()
        {
            personId= this.GetType().Name + (new Random()).Next(1, int.MaxValue);
        }
     }
```

سپس در یک متد مجموعه‌ای از Studentها و teacherها را ایجاد کرده و به لیستی از Personها اضافه میکنم:

```
var student1=new Student(){Name="Iraj",Age=21};
var student1=new Student(){Name="Nima",Age=20};
var student1=new Student(){Name="Sara",Age=25};
var student1=new Student(){Name="Mina",Age=22};
var student1=new Student(){Name="Narges",Age=26};
var teacher1=new Student(){Name="Navaei",Age=45};
var teacher2=new Student(){Name="Imani",Age=50};
```

اما در نهایت اتفاقی که رخ میداد این بود که PersonId همه Studentها یکسان می‌شد ولی قضیه به همین جا ختم نشد؛ وقتی خط به خط برنامه را Debug و مقادیر را Watch می‌کردم، مشاهده می‌کردم که PersonId به درستی ایجاد می‌شود.
در فیزیک نوین اصلی هست به نام عدم قطعیت هایزنبرگ که به زبان ساده میتوان گفت نحوه رخداد یک اتفاق، با توجه به وجود یا عدم وجود یک مشاهده‌گر خارجی نتیجه‌ی متفاوتی خواهد داشت.
کم کم داشتم به وجود قانون مشاهده‌گر در برنامه نویسی هم ایمان پیدا میکردم که این کد فقط در صورتیکه آنرا مرحله به مرحله بررسی کنم جواب خواهد داد!
جالب اینکه زمانیکه personId را نیز ایجاد میکردم، یک دستور برای دیدن خروجی نوشتم مثل این

```
public class Person
    {
        public Person()
        {
            personId= this.GetType().Name + (new Random()).Next(1, int.MaxValue);
            Debug.Print(personId)
        }
     }
```

در این حالت نیز دستورات درست عمل میکردند و personId متفاوتی ایجاد می‌شد! قبل از خواندن ادامه مطلب شما هم کمی فکر کنید که مشکل کجاست؟
این مشکل ربطی به قانون مشاهده‌گر و یا دیگر قوانین فیزیکی نداشت. بلکه بدلیل سرعت بالای ایجاد وهله ها(instance) از کلاسی‌های مطروحه (مثلا در زمانی کمتر از یک میلی ثانیه) زمانی در بازه یک کلاک CPU رخ می‌داد.

هر نوع ایجاد کندی (همچون نمایش مقادیر در خروجی) باعث می‌شود کلاک پردازنده نیز تغییر کند و عدد اتفاقی تولید شده فرق کند.
همچنین برای حل این مشکل میتوان از کلاس تولید کننده اعداد اتفاقی، شبیه زیر استفاده کرد:

```
using System;
using System.Threading;

public static class RandomProvider
```

```
{
    private static int seed = Environment.TickCount;

    private static ThreadLocal<Random> randomWrapper = new ThreadLocal<Random>(() =>
        new Random(Interlocked.Increment(ref seed))
    );

    public static Random GetThreadRandom()
    {
        return randomWrapper.Value;
    }
}
```

نظرات خوانندگان

نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۳/۰۷/۲۷ ۱۴:۴

به جای کلاس Random از جایگزین بهتر آن RNGCryptoServiceProvider استفاده کنید و دوباره برنامه رو تست کنید.