# 随机数

**RNGCryptoServiceProvider 类**

**.NET Framework 2.0**

[其他版本](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

2（共 3）对本文的评价是有帮助 - [评价此主题](http://msdn.microsoft.com/zh-cn/library/system.security.cryptography.rngcryptoserviceprovider(v=vs.80).aspx#feedback)

使用加密服务提供程序 (CSP) 提供的实现来实现加密随机数生成器 (RNG)。无法继承此类。

**命名空间:**System.Security.Cryptography  
**程序集:**mscorlib（在 mscorlib.dll 中）

[**语法**](javascript:void(0))

C#

[**C++**](http://msdn.microsoft.com/zh-cn/library/system.security.cryptography.rngcryptoserviceprovider(v=vs.80).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-1)

[**VB**](http://msdn.microsoft.com/zh-cn/library/system.security.cryptography.rngcryptoserviceprovider(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[ComVisibleAttribute(true)]

public sealed class RNGCryptoServiceProvider : RandomNumberGenerator

J#

/\*\* @attribute ComVisibleAttribute(true) \*/

public final class RNGCryptoServiceProvider extends RandomNumberGenerator

JScript

ComVisibleAttribute(true)

public final class RNGCryptoServiceProvider extends RandomNumberGenerator

[**示例**](javascript:void(0))

下面的代码示例演示如何用 **RNGCryptoServiceProvider** 类创建随机数。

C#

[**C++**](http://msdn.microsoft.com/zh-cn/library/system.security.cryptography.rngcryptoserviceprovider(v=vs.80).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-4)

[**VB**](http://msdn.microsoft.com/zh-cn/library/system.security.cryptography.rngcryptoserviceprovider(v=vs.80).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

//The following sample uses the Cryptography class to simulate the roll of a dice.

using System;

using System.IO;

using System.Text;

using System.Security.Cryptography;

class RNGCSP

{

// Main method.

public static void Main()

{

// Roll the dice 30 times and display

// the results to the console.

for(int x = 0; x <= 30; x++)

Console.WriteLine(RollDice(6));

}

// This method simulates a roll of the dice. The input parameter is the

// number of sides of the dice.

public static int RollDice(int NumSides)

{

// Create a byte array to hold the random value.

byte[] randomNumber = new byte[1];

// Create a new instance of the RNGCryptoServiceProvider.

RNGCryptoServiceProvider Gen = new RNGCryptoServiceProvider();

// Fill the array with a random value.

Gen.GetBytes(randomNumber);

// Convert the byte to an integer value to make the modulus operation easier.

int rand = Convert.ToInt32(randomNumber[0]);

// Return the random number mod the number

// of sides. The possible values are zero-

// based, so we add one.

return rand % NumSides + 1;

}

}

[**继承层次结构**](javascript:void(0))

[System.Object](http://msdn.microsoft.com/zh-cn/library/system.object(v=vs.80).aspx)   
   [System.Security.Cryptography.RandomNumberGenerator](http://msdn.microsoft.com/zh-cn/library/system.security.cryptography.randomnumbergenerator(v=vs.80).aspx)   
    **System.Security.Cryptography.RNGCryptoServiceProvider**

[**线程安全**](javascript:void(0))

此类型的任何公共静态（Visual Basic 中的 **Shared**）成员都是线程安全的，但不保证所有实例成员都是线程安全的。

# 单例模式

第一种最简单，但没有考虑线程安全，在多线程时可能会出问题，不过俺从没看过出错的现象，表鄙视我……

public class Singleton  
{  
    private static Singleton \_instance = null;  
    private Singleton(){}  
    public static Singleton CreateInstance()  
    {  
        if(\_instance == null)

        {  
            \_instance = new Singleton();  
        }  
        return \_instance;  
    }  
}

第二种考虑了线程安全，不过有点烦，但绝对是正规写法，经典的一叉

public class Singleton  
{  
    private volatile static Singleton \_instance = null;  
    private static readonly object lockHelper = new object();  
    private Singleton(){}  
    public static Singleton CreateInstance()  
    {  
        if(\_instance == null)  
        {  
            lock(lockHelper)  
            {  
                if(\_instance == null)  
                     \_instance = new Singleton();  
            }  
        }  
        return \_instance;  
    }  
}  
  
第三种可能是C#这样的高级语言特有的，实在懒得出奇

public class Singleton  
{

    private Singleton(){}  
    public static readonly Singleton instance = new Singleton();  
}

# internal (C# Reference)

The **internal** keyword is an access modifier for types and type members. Internal types or members are accessible only within files in the same assembly.

Types or members that have access modifier protected internal can be accessed from the current assembly or from types that derived from the containing class.

# Static Constructors (C# Programming Guide)

A static constructor is used to initialize any static data, or to perform a particular action that needs to be performed once only. It is called automatically before the first instance is created or any static members are referenced.

Static constructors have the following properties:

* A static constructor does not take access modifiers or have parameters
* A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.
* A static constructor cannot be called directly.
* The user has no control on when the static constructor is executed in the program.
* A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.
* Static constructors are also useful when creating wrapper classed for unmanaged code, when the constructor can call the LoadLibrary method.
* If a static constructor throw an exception, the runtime will not invoke it a second time, and the type will remain uninitialized for the lifetime of the application domain in which your program is running.