readonly只能在声明初始化或构造函数中初始化。

readonly可以是实例域也可以是静态域。

readonly的类型可以是C#语言的任何类型

readonly在运行时计算其值。

readonly只能修饰类的成员。

const常量隐含为静态变量（这与C++不同），不能再用static修饰。

const常量必须在声明时赋值，并且要求编译器能够在编译时期计算出确定的值。

const可以修饰数值类型（如sbyte,byte,int,bool decimal,double等），可以修饰string，enum类型，或者可以转换为上述类型的类型。不能修饰struct。

const也可以修饰引用类型，但除了string类型之外，其余的引用类型的值都必须为null。这没什么意义。

const既可以修饰类的成员，也可以修饰函数内部的局部变量。

例：程序readonly\_const\_test1

class MyClass {}

class Test

{

static readonly MyClass r1 = new MyClass();

// const MyClass c1 = new MyClass(); // error

const MyClass c1 = null;

const string c2 = "abc";

}

注意：readonly修饰引用变量时，不能将其指向新的对象，如：

r1 = new MyClass(); // error

但如果MyClass有字段存在，可以通过r1修改字段的值。如

r1.Name = “Tom”;

const和readonly的区别：

例：程序readonly\_const\_test1

class Readonly\_ConstTest

{

static readonly int A = B \* 10;

static readonly int B = 10;

const int C = D \* 10;

const int D = 10;

static void Main(string[] args)

{

Console.WriteLine("A is {0}, B is {1}", A, B);

Console.WriteLine("\nC is {0}, D is {1}", C, D);

}

}

输出结果为：

A is 0, B is 10

C is 100, D is 10

原因分析：

A和B均为static readonly，在运行时计算其值，初始均默认为0，在计算A时，B的值为0，所以A为0，B为10。如果将B的定义放在A之前，则A为100，B为10.

C和D均为const，在编译时就计算好了值，所以C为100，D为10.

生成一个dll文件，dll包含一个readonly和一个const：

例：程序readonly\_const\_dll

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace readonly\_const\_dll

{

public class ReadonlyConst

{

public static readonly double pi = 3.14;

public const double PI = 3.14;

}

}

例：程序readonly\_const\_test2

首先引用readonly\_const\_dll.dll

// Copyright 2016.刘珅珅

// author：刘珅珅

// readonly与const区别

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using readonly\_const\_dll;

namespace readonly\_const\_test2

{

class Readonly\_ConstTest

{

static void Main(string[] args)

{

Console.WriteLine("readonly pi " + ReadonlyConst.pi);

Console.WriteLine("const pi " + ReadonlyConst.PI);

Console.ReadKey();

}

}

}

输出为：

readonly pi 3.14

const pi 3.14

然后修改readonly\_const\_dll.dll如下：

public class ReadonlyConst

{

public static readonly double pi = 3.1415;

public const double PI = 3.1415;

}

将编译好的dll文件直接拷贝到之前readonly\_const\_test2.exe的目录下，不重新编译readonly\_const\_test2.exe，然后直接双击运行readonly\_const\_test2.exe，结果如下：

readonly pi 3.1415

const pi 3.14

从两次运行的结果可以看出，const变量是在编译时确定值的，如果修改了const初始化的值，就必须重新编译所有使用const的cs文件，而readonly是在运行时计算值的，如果更改了readonly的值，可以只编译其所在的类库工程，其他地方不需要重新编译。