7/11

笔记本: 暑期实习
创建时间: 2021/7/11 7:55 **更新时间:** 2021/7/12 18:28
作者: 134exetj717
URL: https://www.runoob.com/csharp/csharp-program-structure.html

---

- 创建C#的控制台应用程序
- 了解各类文件的作用

    1. sln 是解决方案文件：作用是打开程序的执行环境，自动调用程序的相关文件，提供一个集成环境
    2. 文件里的第一层目录是执行空间，文件名的文件夹中的内容是系列空间文件，全部用于提供一个集成环境。

- c#命名规范：
    - 仅第一个字母大写，其余小写
    - 不要使用匈牙利命名法，即不用下划线
- C#和C、C++的区别

```
using System;
namespace HelloWorldApplication
{
   class HelloWorld
   {
      static void Main(string[] args)
      {
         /* 我的第一个 C# 程序*/
         Console.WriteLine("Hello World");
         Console.ReadKey();
      }
   }
}
```

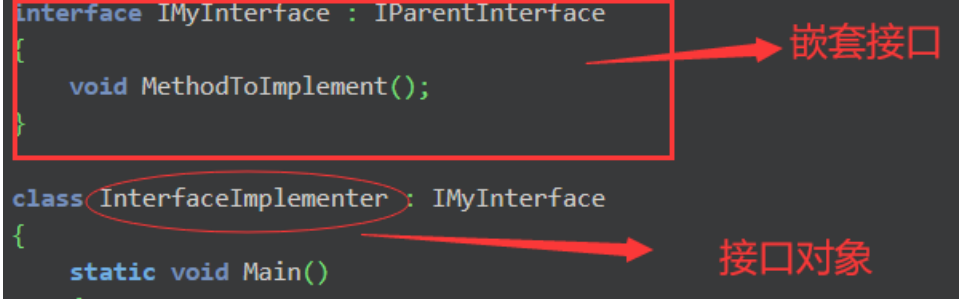
1. C#是基于C、C++创建出来的，集合了两者的优点
2. C#的头文件用using进行引用，同时C#编写一个代码相当于创建了一个命名空间，因此用class+文件名展开，执行文件的展开从main()函数开始
3. 可空类型：？ 表示内容可空；？？ 表示当内容为空时，赋值 num ?? value
4. console.writeLine()表示输出；console.readLine()表示输入
5. 类型转换：（需要 convert.），或者用类似于int.parse（）
    1. toboolean转换为布尔型，（C#不支持int隐式转换为bool，或者将bool转换为Int），显示转换后，默认0、1
    2. tobyte转换为字节类型
    3. tochar转换为单个Unicode类型
    4. todatetime转换为日期-时间结构
    5. todecimal转换为十进制类型
    6. todouble转换为双精度浮点型
    7. toint16、32，64
    8. tosbyte转换为有符号字节类型
    9. tosingle转换为小浮点数类型
    10. tostring转换为字符串
    11. toType转换为指定类型
    12. toUint16、32、61
6. 常量：

    1. @""，表示字符串常量，可以将\当作一个字符处理
7. 封装：
    1. public：公开
    2. private：私有
    3. proceted：保护，仅子类可以看见
    4. internal：同一个程序集可以看到
    5. protected internal：并集，满足其中一个就可以访问
    6. 比如，A是父亲，B、C是孩子，D是母亲，E是私生子，公开是全世界都知道，私有是只有A看到，保护是A、B、C、E都可以看到，interval是A、B、C、D可以看到，最后一个是五个人都能看到
8. 参数传递
    1. 按值传递
    2. 按引用传递
    3. 按输出传递（定义out）
9. c#中提供了string类的各个方法

```
int a, b;
a = int.Parse(Console.ReadLine());
b = int.Parse(Console.ReadLine());

Console.WriteLine($"{a} + {b} = {a+b}");
```

    1. $符号的使用：
    2. compare 比较两个指定的string对象
    3. concat 可以连接两个或三个字符串
    4. contains 返回Bool值，表示指定string对象是否出现在字符串中
    5. copy 将指定string字符串复制给目标string字符串
    6. endswith 返回Bool值，判断指定string对象的结尾是否匹配指定的字符串
    7. equals 返回bool值，判断当前对象是否与指定的string对象具有相同的值
    8. ...
    9. 用于转化值的格式化方法
10. 结构（struct）：可定义有参构造函数，没有析构函数
11. enum，枚举，允许用符号代表数据，默认第一个数据为0：使用，枚举名 = 枚举空间.枚举名
12. 类：静态（无论创建了多少个实例，只有该静态成员的副本）、析构函数（不返回值，不带有任何参数，不能继承和重载）、构造函数；C#跟C++不太一样，它创建类必须要用new
13. 继承：单一继承和多重继承，class 子类名：父类名[,父类名]
14. 抽象类（abstract）和虚方法（virtual）
15. console类，静态类，一个工具，里面包含了writeLine,readKey（类似于system的pause功能）,readLine等函数
16. 不支持宏定义，只支持const int name=数字
17. 装箱操作：将一个值类型的数据隐式地转换成一个对象类型的数据，比如：将整型转换为对象类型；拆箱操作：将一个引用类型显式地转换成一个值类型数据。
18. 多态性：运算符的重载
19. 接口：接口和抽象类非常类似，它本身不具备任何功能或者信息，而是给出了一个标准，该接口的对象必须要遵循这个标准，给出一系列行为、方法。

```
using System;

interface IParentInterface
{
    void ParentInterfaceMethod();
}

interface IMyInterface : IParentInterface
{
    void MethodToImplement();
}

class InterfaceImplementer : IMyInterface
{
    static void Main()
    {
        InterfaceImplementer iImp = new InterfaceImplementer();
        iImp.MethodToImplement();
        iImp.ParentInterfaceMethod();
    }

    public void MethodToImplement()
    {
        Console.WriteLine("MethodToImplement() called.");
    }

    public void ParentInterfaceMethod()
    {
        Console.WriteLine("ParentInterfaceMethod() called.");
    }
}
```



20. 命名空间，namespace 空间名{ }；using 空间名；命名空间嵌套
21. 交错数据[][]，返回的长度是有几个数组；多维数组[,]，返回的长度是总元素个数
22. 预处理指令

预处理器指令	描述
#define	它用于定义一系列成为符号的字符。
#undef	它用于取消定义符号。
#if	它用于测试符号是否为真。
#else	它用于创建复合条件指令，与 #if 一起使用。
#elif	它用于创建复合条件指令。
#endif	指定一个条件指令的结束。
#line	它可以让您修改编译器的行数以及（可选地）输出错误和警告的文件名。
#error	它允许从代码的指定位置生成一个错误。
#warning	它允许从代码的指定位置生成一级警告。
#region	它可以让您在使用 Visual Studio Code Editor 的大纲特性时，指定一个可展开或折叠的代码块。
#endregion	它标识着 #region 块的结束。

- 创建类的实例：

```
类名    对象名=new 类名（）
```

- 访问对象的字段

```
对象名.字段名
```

- 调用对象的方法

```
对象名.方法名（参数表）
```

- 内存分配方式
  - 对象应用变量是一个引用类型的变量，和值类型变量一样，对象引用变量的空间也是在栈空间中分配的。
  - 对象引用实例则是在堆空间中分配的
- 命名空间概述
  - 类是通过命名空间来组织。命名空间提供了可以将类分成逻辑组的方法。
  - 在每个命名空间中，所有的“类”都是独立的
- 使用命名空间
  - 明确指出命名空间的位置
  - 用using关键字引用命名空间

```
using [别名=] 命名空间
```

- 构造函数：重载，默认调用，不能继承，结构体中有但是不能无参
- 析构函数：自动销毁，结构体没有
- 静态字段（字段应该是私有的，这样做是为了保护数据的安全性）
  - 类中所有对象共享的成员，因此不同实例都会直接影响静态字段
  - 对象存在时才有意义

```
static int c
类名.静态方法名（参数表）
```

- 属性
  - 对象的具体特性
  - 不直接操作类的字段，而是通过访问器进行访问
  - 一个属性最多只有一个get访问器和set访问器
  - 相当于对某个属性进行了读写的保护
  - 规范的角度来讲，只出现set是不规范的，其余都可以：因为若只能写，而不能读，那么该字段就丧失本身存在的意义

```
int x,y;    //定义了一个私有字段
public int px{    //通过赋予的属性px来访问私有字段 x
    get{    //get访问器
        return x;}
    set     //set访问器
    { x = value;}
}
```

  - 自动实现的属性：只声明属性而不定义其后备字段（可读可写）

```
public int f
{ set;get;}    //自动实现的属性
```