## 可空类型

在一个正常取值范围内 增加了一个null的取值

比如 在进行投票时 假如就只有两个选项a和b 完全可以使用一个bool值来表示投票结果

但是可能会出现弃权的情况 仅仅使用真假值不够表示投票结果

但是可以增加一个可空值 即 true false null

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Text : MonoBehaviour {

    //可空值类型 new初始化后 变量为空
    int? num1=new int?();
    int? num2;
    double? num3 = new double?();
    double? num4 = 3.14159;
    bool? boolval = new bool?();
    int? num5 = 45;
    int? a = null;//int a不可以为空

    //引用类型x实例化 x不为空
    GameObject x = new GameObject();

    void Start () {
        Debug.Log(num1);//null
        Debug.Log(num2);//null
        Debug.Log(num3);//null
        Debug.Log(num4);//3.14159
        Debug.Log(num5);//45
        Debug.Log(boolval);//null
        Debug.Log(a);//null

        num1 = 100;
        Debug.Log(num1);//100
    }

}
```

```
33
```

## C#预编译指令（预处理器指令）

预编译指令是指导编译器在实际编译开始之前对信息进行预处理

# 预处理器指令符号

一行上 只有空白字符可以出现在预处理器指令之前

在C#中 预处理器指令用于在条件编译中起作用

#define 预处理器

#if 条件指令

```
1  #define A //定义符号常量 必须在文件头写
2
3  using System.Collections;
4  using System.Collections.Generic;
5  using UnityEngine;
6
7  public class Text : MonoBehaviour {
8
9      void Start () {
10 #if !A
11         Debug.Log(100);
12 #else
13         Debug.Log(200);
14 #endif
15
16 #if (DEBUG&&VC_10)
17         Debug.Log("100");
18 #elif (VC_10||!UNITY_ANDROID)
19         Debug.Log("100");
20 #else
21
22 #endif
23
24     }
```

```
25
26      #region 方法
27      void fun()
28      {
29      }
30      void fun1()
31      {
32      }
33      #endregion
34  }
35
```

***C#指针**

C#不安全代码

当一个代码块使用关键字unsafe标记时

C#会允许函数中使用指针变量

不安全代码 非托管代码 是指使用了指针变量的代码块

在C#使用指针

控制台: 项目->属性->生成->允许不安全代码->打勾

Unity:  在项目路径中创建smcs.rsp文件 添加一行文字：-unsafe

```
1   static void Main(string[] args)
2   {
3               unsafe
4               {
5                   int a=100;
6                   int* p = &a;
7                   Console.WriteLine((int)p);
8                   Console.WriteLine(*p);
9               }
10  }
11
12  unsafe void Fun()
```

```
{
            int a = 100;
            int* p = &a;
            Console.WriteLine((int)p);
            Console.WriteLine(*p);
}
```

## C#命名空间

C#中一种代码组织的形式

把代码通过命名空间的方式进行分类和组织

不同的人在设计程序时 可能会出现相同的变量名或类型名 使用命名空间最直观的目的 就是避免冲突

```
using System.Collections;
using System.Collections.Generic;//.NET提供的泛型命名空间
using UnityEngine;//Unity引擎的命名空间
using UnityEditor;//Unity编辑器的命名空间
using System;//系统命名空间
using MySpace1;//引用自定义的命名空间MySpace1
using MySpace2;

//自定义的命名空间MySpace1
namespace MySpace1
{
    public class ClassA
    {

    }

}

//自定义的命名空间MySpace2
namespace MySpace2
{
    public class ClassA
    {

```

```
    }

}

public class Text : MonoBehaviour {

    void Start () {
        //通过命名空间索引 访问MySpace1空间里的类型
        MySpace1.ClassA a1 = new MySpace1.ClassA();
        MySpace2.ClassA a2 = new MySpace2.ClassA();

        //文件头添加命名空间指令后 可以直接访问类型
        //但是如果同时出现多个命名空间都包含相同名字的类型 依然产生冲突
        //必须写明命名空间
        //ClassA b = new ClassA();

        MySpace1.ClassA b1 = new MySpace1.ClassA();
        MySpace2.ClassA b2 = new MySpace2.ClassA();

        //在Unity开发中 需要注意的类型 随机数
        System.Random random = new System.Random();//C#系统提供的随机数类型
        int x = random.Next(100);

        UnityEngine.Random.Range(100, 200);//Unity提供的随机数类型

        //不明确引用 C#和Unity都提供了一个Object类型
        //C#的Object是所有类型的根父类  也是UnityEngine.Object的根父类
        object b = 100;
        UnityEngine.Object x;

        Fun(transform);
    }

    void Fun(params UnityEngine.Object[] datas)
    {

    }
}

```

## C#类库

类库(class library) 面向对象的可重用类型集合 包含 类 接口 抽象类

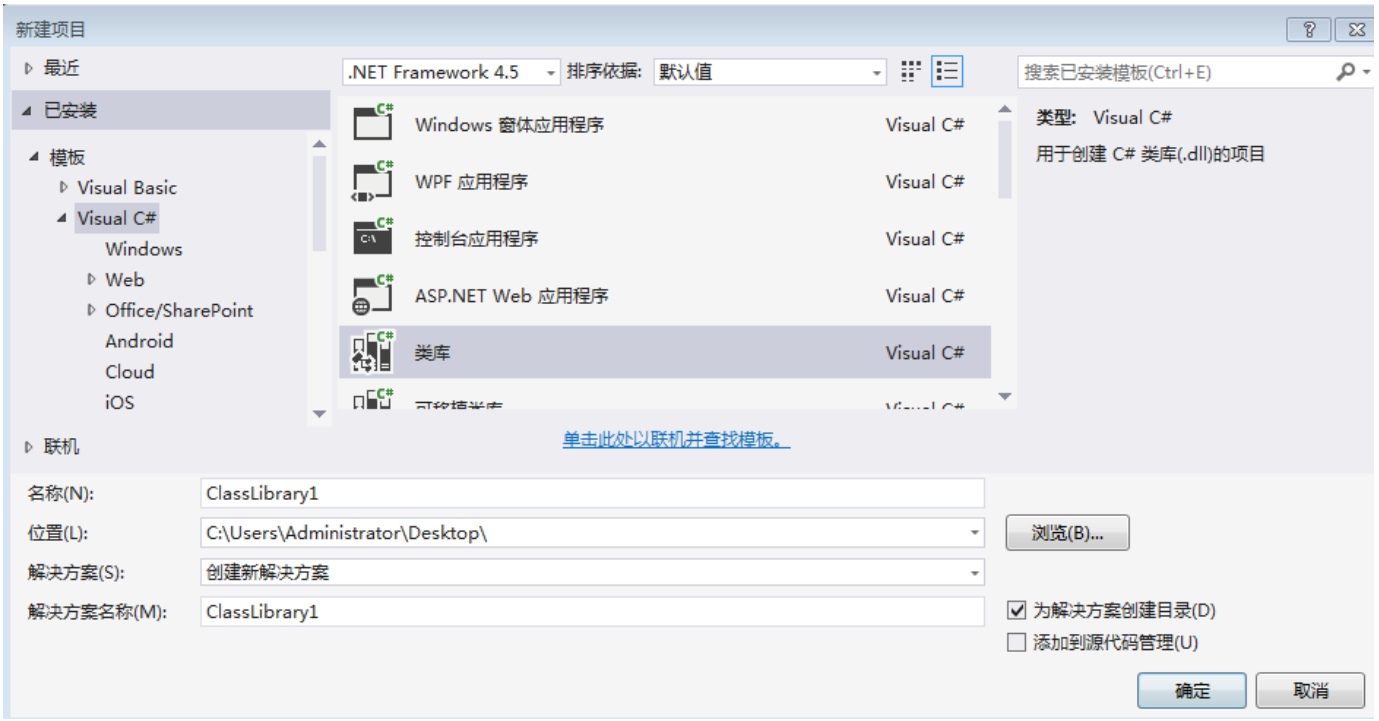
一般来说 类库是为具体项目服务的

例如:LitJson 类库 封装了解析和构建Json的需要的具体的类型和函数方法

LitJson类库可以应用在任何项目中

流程

使用VS创建类库项目




类库默认生成的类：

注意：类库是不需要单独执行的 所以和控制台项目不同 类库项目不包含Main方法

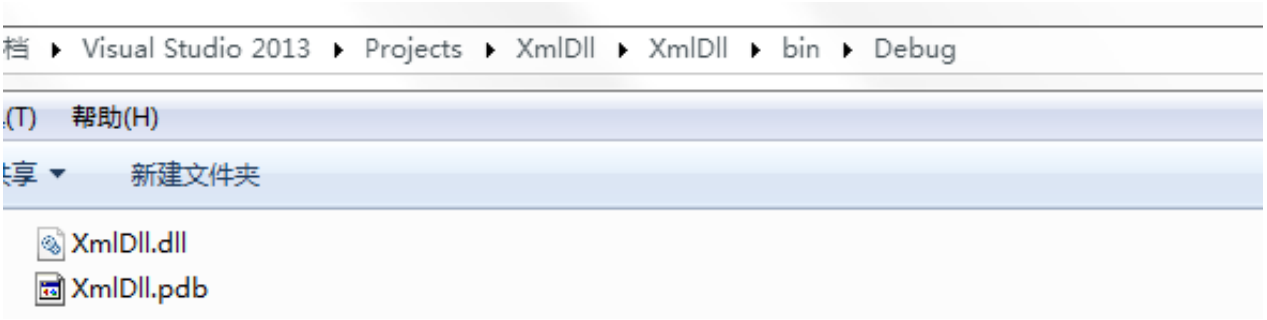
```
1
using System;
2
using System.Collections.Generic;
3
using System.Linq;
4
using System.Text;
5

6
namespace XmlDll
7
{
```

```
    public class Class1
    {
        public int Add(int a,int b)
        {
            return a + b;
        }

    }
}

```

添加新的类：

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UnityEngine;

namespace XmlDll
{
    //Config类前加public 访问权限 保证其他项目可以继承于此类
    public class Config
    {
        public virtual void ReadLoad()
        {
            Config2 config2 = new Config2();
        }
    }

    //internal 访问修饰符
    //config2类只能在本类库中访问
    internal class Config2
    {

    }
}

```

生成类库项目 得到dll文件

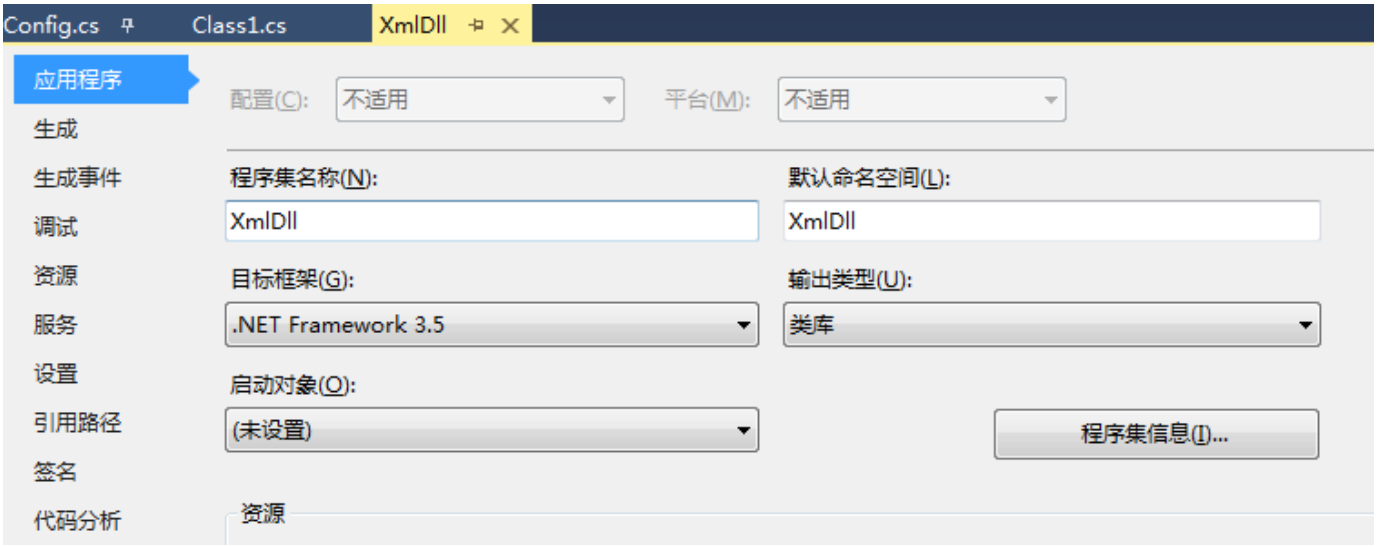



把文件复制到Unity项目中

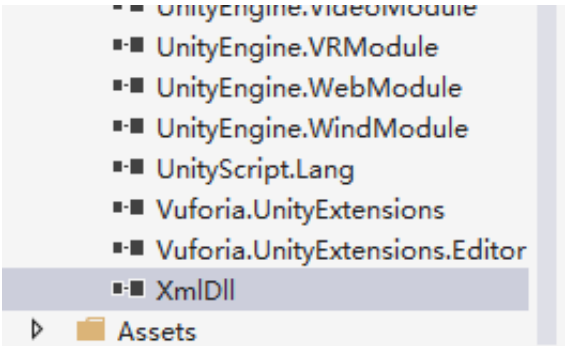
可能会弹出的错误

原因：Unity编辑器版只支持.NET3.5 所以类库的目标框架需要修改

项目->属性->目标框架




如果出现引用失败的情况 可以手动添加引用




在Unity中使用类库的类型和调用方法

```
1 using System.Collections;
```

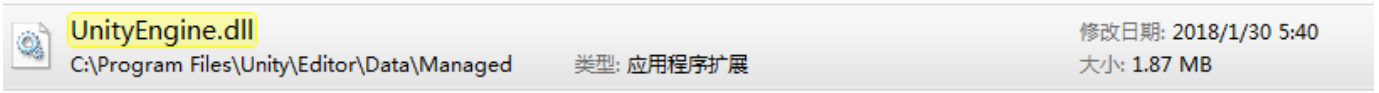
```
2  using System.Collections.Generic;//.NET提供的泛型命名空间
3  using UnityEngine;//Unity引擎的命名空间
4  using XmlDll;//引入自定义类库的命名空间
5
6
7  //继承于类库中的基类 并重写基类的虚方法
8  public class ClassB:Config
9  {
10     public override void ReadLoad()
11     {
12         base.ReadLoad();
13     }
14 }
15 public class Text : MonoBehaviour {
16
17     void Start () {
18         Class1 x = new Class1();
19         Debug.Log(x.Add(100, 200));
20     }
21 }
22
```

在类库中使用Unity的API

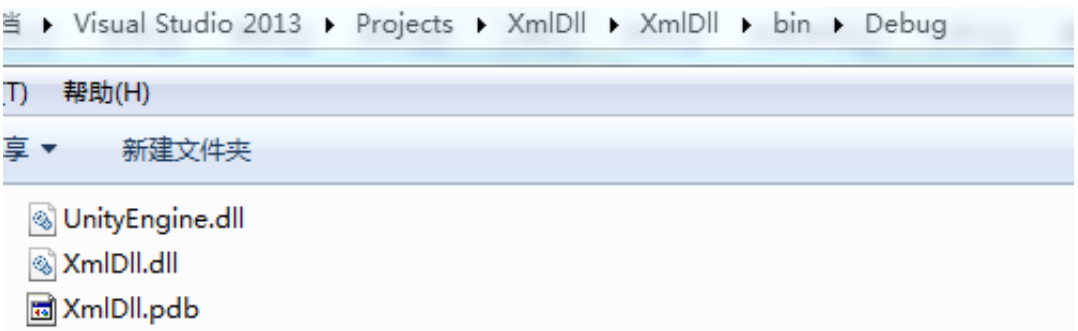
比如需要在类库中添加一个继承于MonoBehaviour的类

就需要类库引用UnityEngine.dll

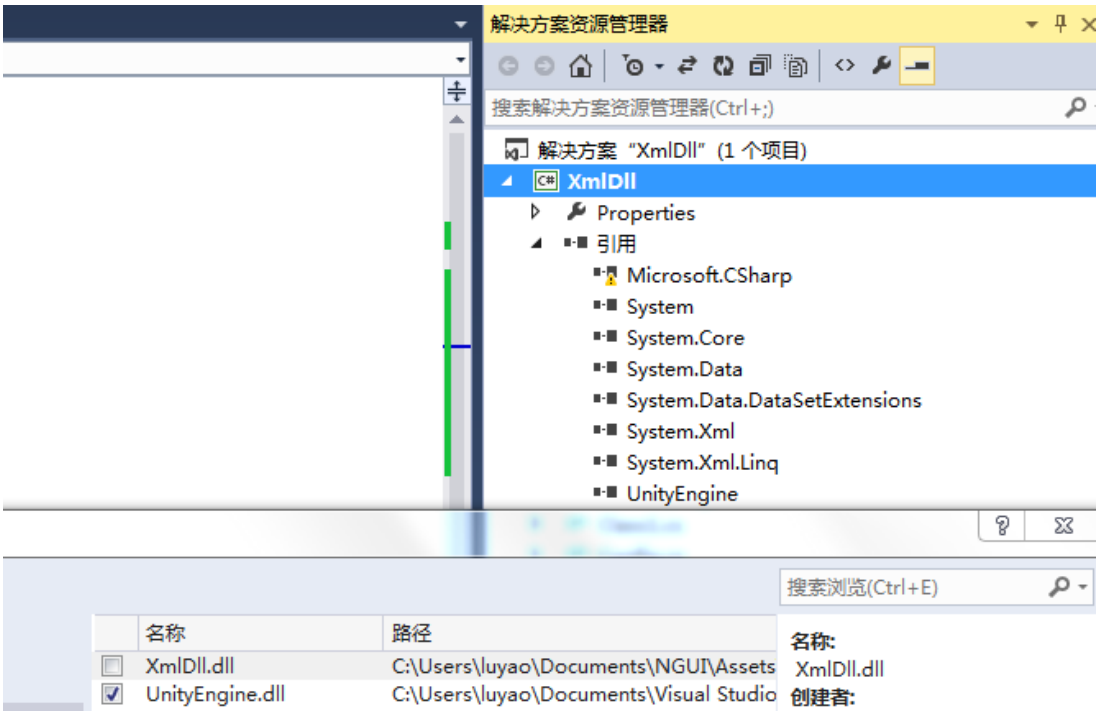
在Unity安装目录中搜索UnityEngine.dll




复制到类库项目中



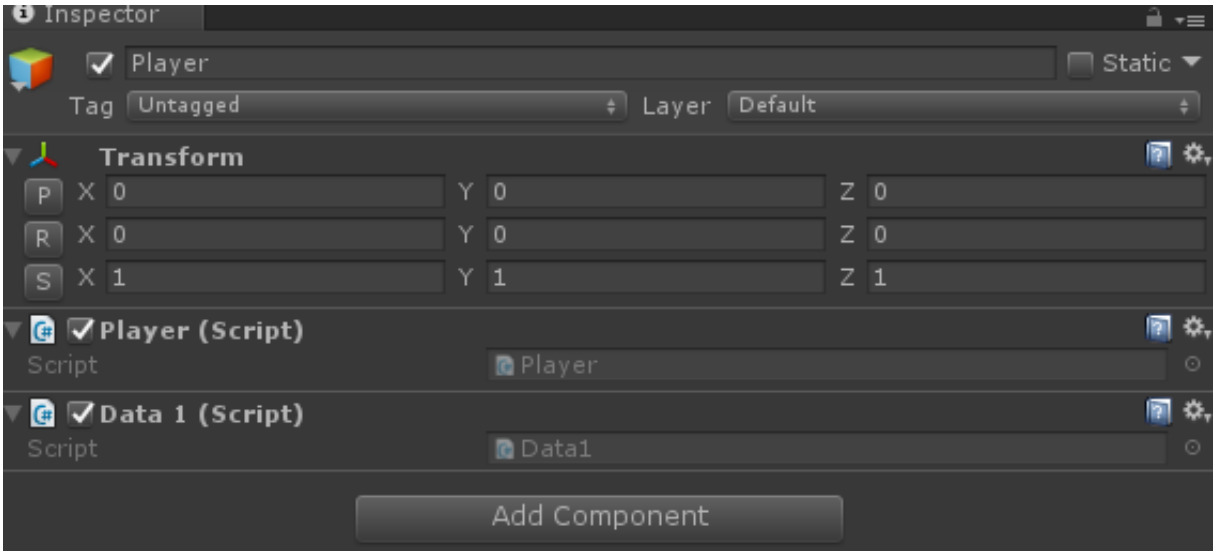

类库项目中添加引用




类库中创建新的类 继承于Mono 重新生成复制到Unity项目中

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UnityEngine;

namespace XmlDll
{
    public class Data1:MonoBehaviour
    {
        void Start()
        {
            Debug.Log("Start");
        }
        void Update()
        {
            Debug.Log("Update");
        }
    }
}

```

在游戏物体上绑定类库中的Data1类型 就可以使用Unity的生命周期




在项目开发时 需要公共的框架可以选择在类库项目中实现 然后在Unity项目中引用