

委托的概念，重要性，自定义委托的声明，委托的历史：函数指针，创建实例，什么是目标方法，单播委托和多播委托，委托的缺点，Action委托和Func委托

---

委托的定义

## C# 委托 (Delegate)

C# 中的委托 (Delegate) 类似于 C 或 C++ 中函数的指针。委托 (Delegate) 是存有对某个方法的引用的一种引用类型变量。引用可在运行时被改变。

委托 (Delegate) 特别用于实现事件和回调方法。所有的委托 (Delegate) 都派生自 System.Delegate 类。

## C# 事件 (Event)

事件 (Event) 基本上说是一个用户操作，如按键、点击、鼠标移动等等，或者是一些提示信息，如系统生成的通知。应用程序需要在事件发生时响应事件。例如，中断。

C# 中使用事件机制实现线程间的通信。

## 通过事件使用委托

事件在类中声明且生成，且通过使用同一个类或其他类中的委托与事件处理程序关联。包含事件的类用于发布事件。这被称为 发布器 (publisher) 类。其他接受该事件的类被称为 订阅器 (subscriber) 类。事件使用 发布-订阅 (publisher-subscriber) 模型。

发布器 (publisher) 是一个包含事件和委托定义的对象。事件和委托之间的联系也定义在这个对象中。发布器 (publisher) 类的对象调用这个事件，并通知其他的对象。

订阅器 (subscriber) 是一个接受事件并提供事件处理程序的对象。在发布器 (publisher) 类中的委托调用订阅器 (subscriber) 类中的方法（事件处理程序）。

委托的概念

一个 委托 是一种 类, 可以 指向一个或者多个方法 (这个委托有参数列表和返回值类型).

当你 实例化委托后、即创建委托的实例,

这个委托类型的实例, 可以和任何其他方法相关联起来, 即可以存储这些方法的引用[只要]类型兼容(任何方法它们的签名和返回值与委托类型的签名和返回值保持一致, 即类型兼容).

你可以间接调用这些方法, 通过 委托类型的实例(对象).

现实中的委托:

同学要回宿舍了，我通过手机发消息，委托同学帮我拿快递

邻居出门把钥匙给你，委托你，如果邻居妻子没带钥匙，就

帮她开门

---

程序中的委托:

委托

程序世界 (抽象)

```

1 using UnityEngine;
2
3 public class GameManager : MonoBehaviour {
4
5     public delegate void MyDelegate();
6     MyDelegate myDelegate;
7
8     private void Enable() {
9         myDelegate += FindObjectOfType<MainPlayer>().GetExp;
10    myDelegate += FindObjectOfType<MainPlayer>().GetBonus;
11    myDelegate += FindObjectOfType<MainPlayer>().ShowWinCanvas;
12    myDelegate += FindObjectOfType<MainPlayer>().GetExp;
13 }
14
15 public void OnGameOver() {
16     FindObjectOfType<MainPlayer>().GetExp();
17     FindObjectOfType<MainPlayer>().GetBonus();
18     FindObjectOfType<UIManager>().ShowWinCanvas();
19     FindObjectOfType<AnimationManager>().AnimationPlay();
20 }

```

```

1 using UnityEngine;
2 using UnityEngine.UI;
3
4 public class UIManager : MonoBehaviour {
5
6     public GameObject winPanel;
7
8     public void ShowWinCanvas() {
9         winPanel.SetActive(true);
10 }

```

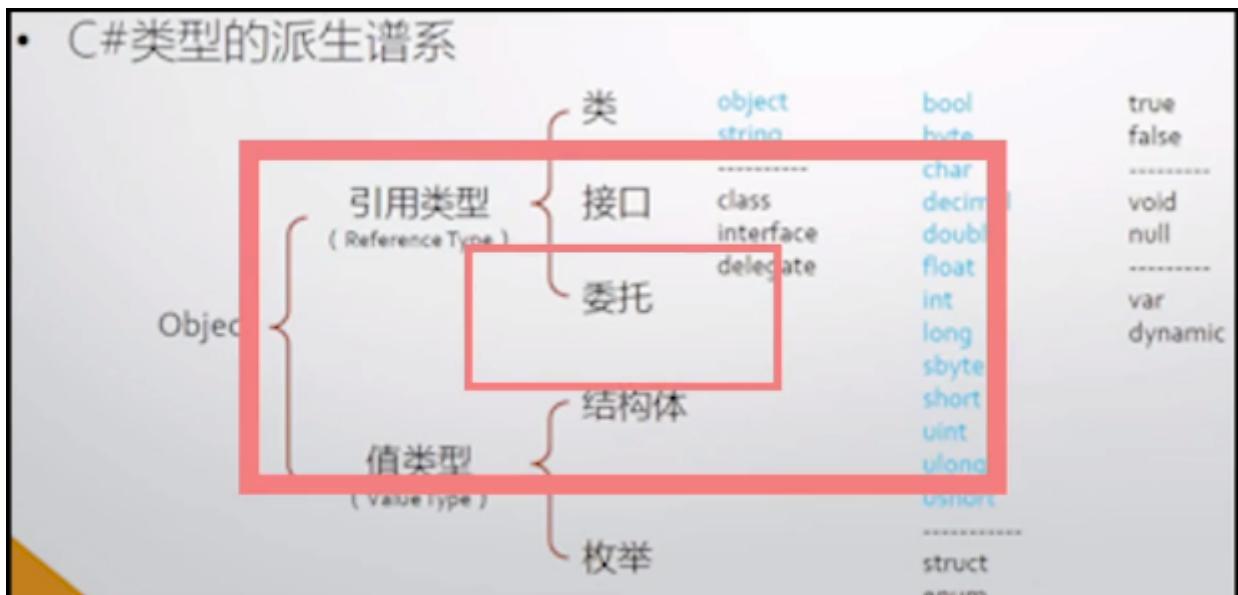
```

1 using UnityEngine;
2
3 public class MainPlayer : MonoBehaviour {
4
5     public int exp, bonus;
6     public int expValue, bonusValue;
7
8     public void GetExp() {
9         exp += expValue;
10 }
11
12     public void GetBonus() {
13         bonus += bonusValue;
14 }
15 }

```

本人不需要亲自执行 全权委托第三方，替他执行具体事务

### 委托的类型



委托是一种类，是引用类型

这个委托类型可以指向任何方法

只要这个方法的【参数类型】和【返回值类型】  
同我们声明的这个【委托类型】保持绝对的类型兼容

## 委托可能导致「内存泄漏 Memory Leak」

多播委托的缺点：

```
//多播委托  
myDelegate += ChangeColor;  
myDelegate += Log;  
}
```

委托会引用一个方法，这个方法是实例方法的话（非静态方法），实例方法也就是说它隶属于一个对象，实例的方法不属于static方法。如果委托引用这个方法的话，这个对象必须存在内存中，即使没有其他引用变量，引用这个对象，这个对象的内存也不能得以释放，因为一旦释放，委托便不能间接的调用对象的方法，因此委托可能导致内存泄漏-----》程序崩溃

---

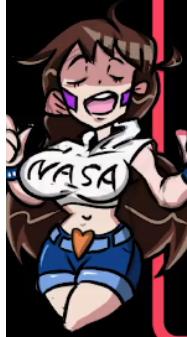
Action 和Func 委托

\_num2;  
**Action委托呢，无返回值、但是可以有参数列表**

**而我们的Func委托呢，有返回值，也可以有参数列表  
(Action和Func都为泛型委托)**

```
Action action01;  
Func<string> func01;  
Func<double, double, double> func02;  
  
private void OnEnable()  
{  
    action01 = new Action(Teleport);  
    func01 = new Func<string>(Log);  
    func02 = new Func<double, double, double>(Add);  
}
```

模板方法 回调方法



## 将委托作为参数传递到方法中 模板方法 回调方法

有一处不确定的,其余代码都是确定好的

这个不确定的部分,就靠我们传进来的  
委托类型的参数所包含的这个方法来“填补”

(Func<T>委托)

以回调方法形式使用【委托】

根据逻辑,动态选择是否调用

(Action<T>委托)