【Unity基础教程】重点知识汇总

（十一）

Unity C#委托与事件

委托是什么（概念）

概念：委托是一个可以引用**一个或多个方法**的**类型（允许你将方法作为参数传递并进行调用）**。它定义了方法的**签名**（参数类型和返回值类型）。在C#中使用**delegate**关键字来声明委托类型。它类似于C/C++中的函数**指针**，但更加安全和灵活。

用途：用于定义**回调机制**，使得方法可以作为参数传递，从而实现动态调用。在事件处理、回调函数和异步编程中非常有用。

***签名：**在C#中，方法签名是指方法的**名称**（唯一标识该方法的字符串）以及其**参数类型**（方法接受的参数类型及其顺序。参数的名称不影响签名，但参数的类型、数量和顺序是非常重要的）的组合。签名定义了方法的输入和输出特性，并用于区分不同的方法。

示例：void PrintMessage(string message);

方法名称：PrintMessage　参数列表：string message（接受一个字符串类型的参数）

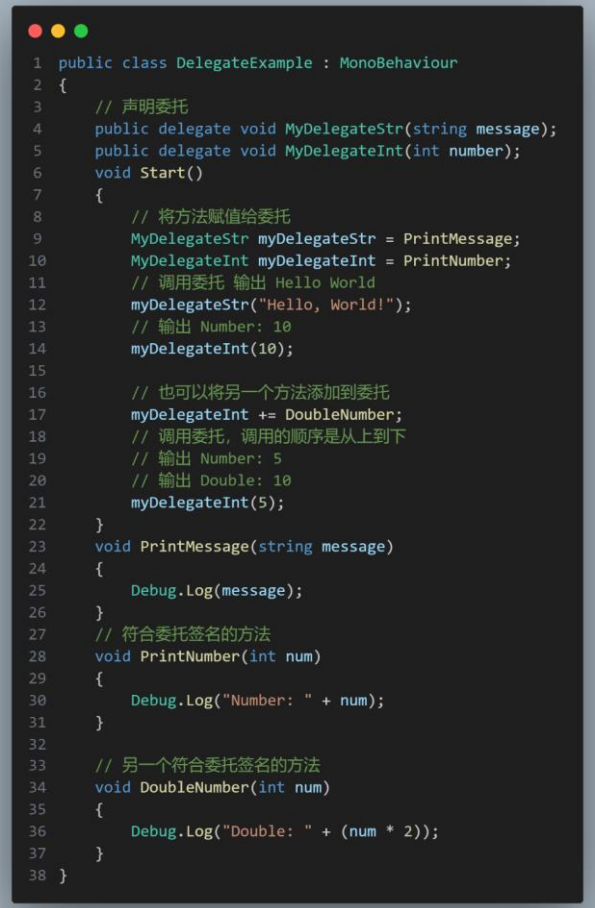
委托的声明与使用（具体实现1）

```
1  public class DelegateExample : MonoBehaviour
2  {
3      // 声明委托
4      public delegate void MyDelegate(string message);
5      void Start()
6      {
7          // 将方法赋值给委托
8          MyDelegate myDelegate = PrintMessage;
9          // 调用委托
10         myDelegate("Hello, World!");
11     }
12     void PrintMessage(string message)
13     {
14         Debug.Log(message);
15     }
16 }
```

语句解释：

- public delegate void MyDelegate(string message); 定义了一个名为MyDelegate的委托类型，它接受一个string（字符串）类型的参数，并且没有返回值。
- MyDelegate myDelegate = PrintMessage;创建一个名为myDelegate的委托实例，并将PrintMessage方法赋值给它。**PrintMessage方法符合MyDelegate的签名，接受一个字符串参数并返回void。**
- myDelegate("Hello, World!");调用委托myDelegate，并传递字符串"Hello, World!" 作为参数。

委托的声明与使用（具体实现2）



```
public class DelegateExample : MonoBehaviour
{
    // 声明委托
    public delegate void MyDelegateStr(string message);
    public delegate void MyDelegateInt(int number);
    void Start()
    {
        // 将方法赋值给委托
        MyDelegateStr myDelegateStr = PrintMessage;
        MyDelegateInt myDelegateInt = PrintNumber;
        // 调用委托 输出 Hello World
        myDelegateStr("Hello, World!");
        // 输出 Number: 10
        myDelegateInt(10);

        // 也可以将另一个方法添加到委托
        myDelegateInt += DoubleNumber;
        // 调用委托，调用的顺序是从上到下
        // 输出 Number: 5
        // 输出 Double: 10
        myDelegateInt(5);
    }
    void PrintMessage(string message)
    {
        Debug.Log(message);
    }
    // 符合委托签名的方法
    void PrintNumber(int num)
    {
        Debug.Log("Number: " + num);
    }

    // 另一个符合委托签名的方法
    void DoubleNumber(int num)
    {
        Debug.Log("Double: " + (num * 2));
    }
}
```

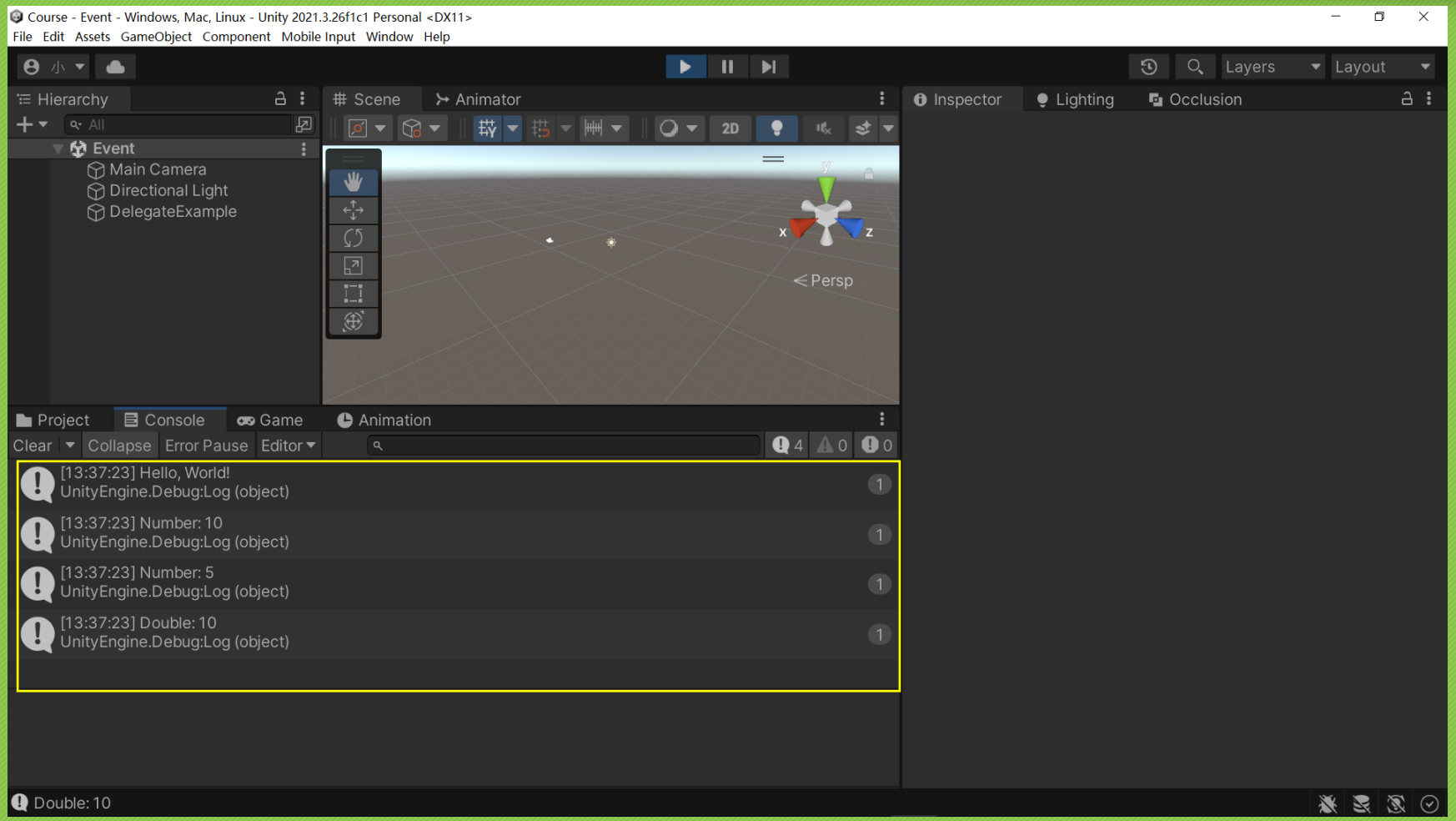
语句解释：

多播委托：通过myDelegateInt += DoubleNumber添加另一个方法。调用myDelegateInt(5)时会先调用PrintNumber方法，然后调用DoubleNumber方法。

补充解释：

虽然在这两个简单场景下直接调用方法似乎更简单，但使用委托可以使代码更具灵活性、可维护性和可扩展性。

委托的声明与使用（运行结果）



事件是什么（概念）

概念：事件是基于委托的一种**高级封装**，可以认为事件是一种委托的**特殊用法（机制）。**在C#中，使用**event**关键字来声明事件，通常基于一个委托类型。委托允许你保存对方法的引用，而事件是进一步**对委托进行封装**，使其只能由事件的声明者触发（只能在事件声明的类内部触发），而外部只能通过**订阅**（+=）或**取消订阅**（-=）事件，无法直接调用事件。

用途：事件通常用于实现脚本之间的**解耦和通信**，通知多个监听者某个特定的事情发生了（它允许一个对象**（发布者）**向多个对象**（订阅者）**发送通知），实现发布-订阅模式，从而避免脚本之间过多的**相互依赖**。

事件的声明与使用（具体实现1）

```
public class EventPublisher : MonoBehaviour
{
    // 声明一个委托类型
    public delegate void MyEventHandler(string message);
    // 基于委托声明事件
    public event MyEventHandler OnEventOccurred;
    private void Start()
    {
        // 订阅事件
        OnEventOccurred += PrintMessage;
        // 触发事件
        TriggerEvent();
    }
    void TriggerEvent()
    {
        // 使用空条件操作符（?.）安全地调用事件。
        // 如果没有任何方法订阅该事件，调用将不会发生。
        OnEventOccurred?.Invoke("Hello, Event!");
    }
    void PrintMessage(string message)
    {
        Debug.Log(message);
    }
}
```

语句解释：

- public event MyEventHandler OnEventOccurred;基于委托MyEventHandler声明了一个事件OnEventOccurred。事件是对委托的封装，允许其他类订阅和响应它。
- 这里将PrintMessage方法订阅到OnEventOccurred事件。当事件被触发时，PrintMessage方法将被调用。
- void TriggerEvent();该方法用于触发OnEventOccurred 事件。使用**空条件操作符**（?.）安全地调用事件。如果没有任何方法订阅该事件，调用将不会发生。

事件的声明与使用（具体实现2）

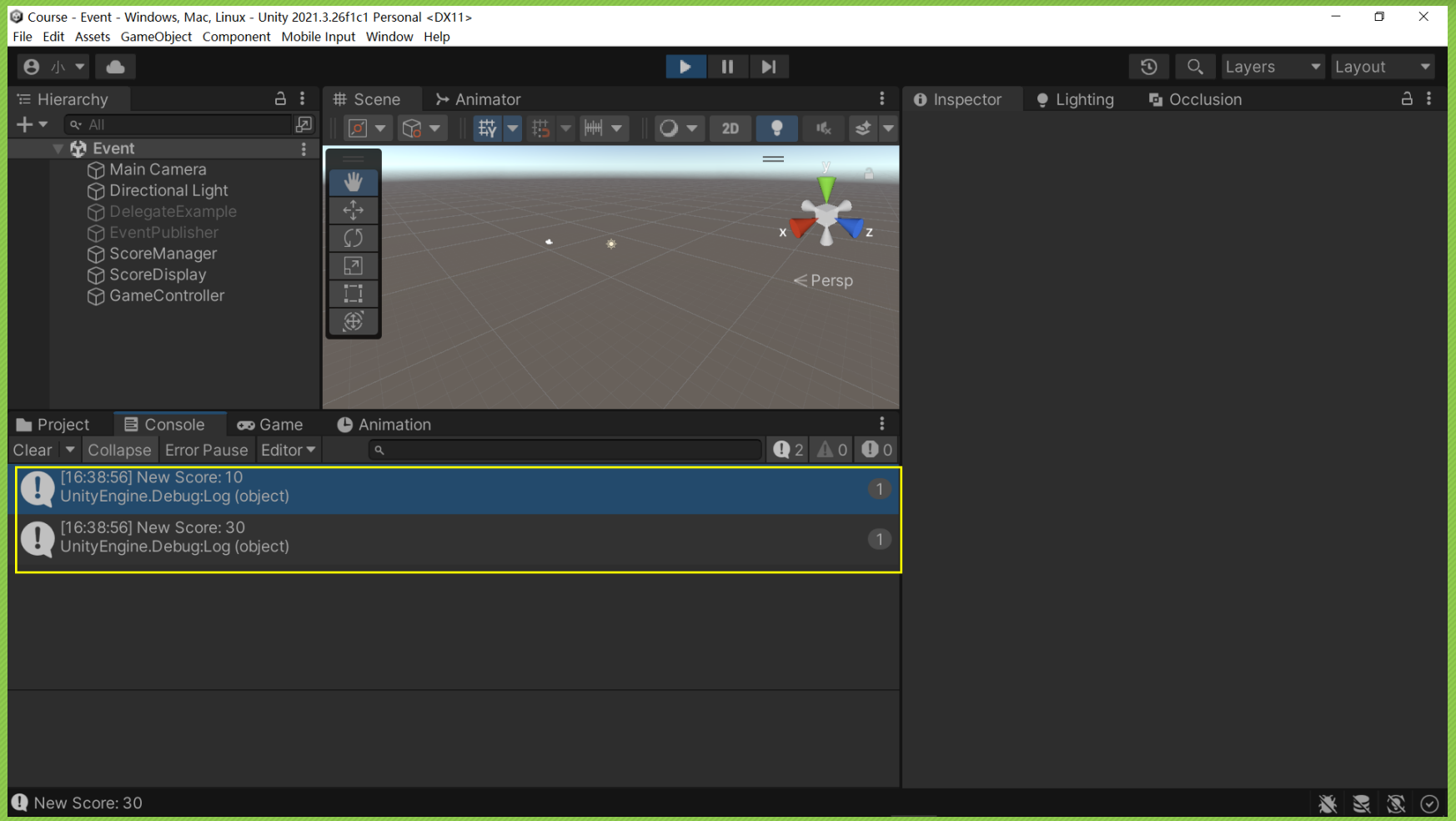
```csharp
// 发布者类
public class ScoreManager : MonoBehaviour
{
    // 定义一个委托
    public delegate void ScoreChanged(int newScore);
    // 声明一个事件
    public event ScoreChanged OnScoreChanged;
    private int score;
    public void AddScore(int points)
    {
        score += points;
        // 触发事件
        OnScoreChanged?.Invoke(score);
    }
}
```

```csharp
// 订阅者类
public class ScoreDisplay : MonoBehaviour
{
    public ScoreManager scoreManager;
    private void OnEnable()
    {
        // 订阅事件
        scoreManager.OnScoreChanged += UpdateScoreDisplay;
    }
    private void OnDisable()
    {
        // 取消订阅事件
        scoreManager.OnScoreChanged -= UpdateScoreDisplay;
    }
    private void UpdateScoreDisplay(int newScore)
    {
        Debug.Log("New Score: " + newScore);
    }
}
```

```csharp
public class GameController : MonoBehaviour
{
    public ScoreManager scoreManager;
    private void Start()
    {
        // 测试增加分数
        // 输出: New Score: 10
        scoreManager.AddScore(10);
        // 输出: New Score: 30
        scoreManager.AddScore(20);
    }
}
```

语句解释：ScoreManager类是一个**发布者**类，负责管理分数，并在分数改变时触发**OnScoreChanged**事件。AddScore方法会增加分数并触发事件。使用?. 操作符确保在事件有订阅者时才调用。ScoreDisplay类是一个**订阅者**类，它在OnEnable中**订阅**OnScoreChanged事件，并在OnDisable中**取消订阅**。UpdateScoreDisplay方法处理分数变更事件。

事件的声明与使用（运行结果）



*事件的声明与使用（简化用法）

```
1  public class EventSimple : MonoBehaviour
2  {
3      // 除了使用自定义委托类型，也可以使用内置的 Action 和 Action<T>
4      public event Action<int> PlayerScoredEvent;
5      private int score = 0;
6      public void PlayerScored()
7      {
8          score += 10;
9          Debug.Log("玩家得分: " + score);
10         // 使用 ?.Invoke 简化触发
11         PlayerScoredEvent?.Invoke(score);
12     }
13 }
```

```
1  public class EventSubscriber : MonoBehaviour
2  {
3      public EventSimple publisher;
4      private void Start()
5      {
6          if (publisher != null)
7          {
8              publisher.PlayerScoredEvent += OnPlayerScored;
9          }
10     }
11     private void OnDestroy()
12     {
13         if (publisher != null)
14         {
15             publisher.PlayerScoredEvent -= OnPlayerScored;
16         }
17     }
18     private void OnPlayerScored(int score)
19     {
20         Debug.Log("订阅者收到分数: " + score);
21     }
22 }
```

委托与事件的区别（总结）

委托与事件的联系与区别

特性	委托	事件
基础类型	委托本身就是一个类型。	事件是基于委托的。
访问限制	可以在类内或类外直接调用委托实例。	只能在事件声明类内触发，外部只能订阅或取消订阅。
发布-订阅模式支持	可以实现，但需要手动管理订阅和取消订阅。	专为发布-订阅模式设计，使用更安全方便。
用法	用于回调函数或多播场景。	用于发布者通知多个订阅者的场景。
灵活性	灵活，但不够安全。	对外部限制更多，但更安全。

总结：委托是方法的引用，可以用于灵活的回调机制，但需要手动管理订阅和取消订阅。事件是基于委托的，是对委托的封装（进一步抽象，可以被视为一种特殊的委托）。提供了更安全的发布-订阅机制，用于实现对象间的通知以及对状态变化的处理。在Unity中大多数系统事件（如按钮点击、触摸输入）都是基于事件的封装。同时，事件的使用方式更为规范，通常包含合适的添加和删除事件处理程序的语法。

【Unity基础教程】重点知识汇总

（十一）

Unity C#委托与事件