【Unity基础教程】重点知识汇总

（十二）

Unity C#接口与抽象类

接口是什么（概念）

概念：接口是一种**完全抽象**的**类型**，是一种定义**契约（规范）**的方式。它规定了类必须要实现的成员（方法、属性、事件等），但**不提供具体实现方法**。在C#中使用**interface**关键字来定义接口，接口的名字通常以“I”开头以示区分（约定俗成）。

特点：

- 接口用于定义类应该实现哪些方法或属性，但并不提供具体实现方法。
- 接口中的所有方法都是抽象的，**没有方法体**。
- 一个类可以实现**多个**接口，这将允许更灵活的设计（用于定义不同类之间**共享**的行为）。
- 接口中的成员默认是**public**的，不能包含private或protected成员。
- 接口中只能包含方法签名、属性、事件和索引器，**不能定义任何字段（变量）**。
- 接口**不能被实例化**。

接口的定义及使用（具体实现）

```
// 定义一个接口
public interface IDriveable
{
    // 定义一个没有实现的Drive方法
    void Drive();
    // 定义一个没有实现的Stop方法
    void Stop();
}

public class Game : MonoBehaviour
{
    void Start()
    {
        // 使用接口类型来引用对象
        IDriveable myCar = new Car();
        // 输出 The car is driving.
        myCar.Drive();
        // 输出 The car has stopped.
        myCar.Stop();
    }
}
```

```
// 实现接口的类
public class Car : IDriveable
{
    public void Drive()
    {
        Debug.Log("The car is driving.");
    }

    public void Stop()
    {
        Debug.Log("The car has stopped.");
    }
}
```

语句解释：

在这个例子中，IDriveable是一个接口，定义了两个方法Drive和Stop，但没有实现它们。然后，Car类实现了这个接口，并提供了方法的具体实现。

接口的优缺点（比较）

优点：

- **灵活性高（支持多重实现）。**一个类可以实现多个接口（用**逗号**隔开），而不像在继承时只能继承一个。
- **解耦设计。**接口定义了一组契约，使得代码对具体实现的**依赖**减少。在上一页IDriveable 示例中，Car的使用者只需依赖接口IDriveable，而需要**不关心**具体类是Car还是其他的实现类（例如Truck或Bike）。

缺点：

- **没有实现的默认行为。**接口中的方法必须由实现类完全定义，无法提供任何默认行为。在IDriveable中Drive和Stop必须由每个实现类单独实现，哪怕这些实现逻辑可能是**重复**的。
- **无状态支持。**接口不能包含字段或状态，只能定义行为。如果需要在IDriveable中定义一个通用的状态，比如IsDriving，你只能通过其他方式实现，而不能直接在接口中定义。

抽象类是什么（概念）

概念：抽象类是一个**不能被实例化的类**，它用于为派生类提供一个**共同的基础**。抽象类可以包含方法的实现，也可以包含抽象方法（没有实现）。抽象类的目的是**共享代码并强制子类实现某些方法**。一个类只能继承一个抽象类。

特点：

- 可以包含方法的实现，也可以包含抽象方法（没有实现）。
- 可以有字段（通过构造函数初始化）、构造函数、属性等。
- 一个类只能继承一个抽象类（**单继承**），用于定义共享代码和提供某些默认行为。
- 抽象类成员中可以有public、protected、private等修饰符。
- 抽象类**不能被直接实例化**，必须通过子类实现。

抽象类的定义及使用（具体实现）

```csharp
// 定义一个抽象类
public abstract class Vehicle
{
    // 字段
    public string Name { get; set; }

    // 构造函数
    public Vehicle(string name)
    {
        Name = name;
    }

    // 定义一个普通方法
    public void Honk()
    {
        Debug.Log("Beep! Beep!");
    }

    // 定义一个抽象方法
    public abstract void Move();
}
```

```csharp
public class CarAbs : Vehicle
{
    // CarAbs类的构造函数调用了基类（子类）Vehicle的构造函数，通过base(name)名称。
    public CarAbs(string name) : base(name) { }

    // 实现抽象方法
    public override void Move()
    {
        Debug.Log(Name + " is moving.");
    }
}
```

```csharp
public class Game : MonoBehaviour
{
    void Start()
    {
        // 定义了一个Vehicle类型的变量myCar，并实例化为Car类型，名称为"Toyota"
        Vehicle myCar = new CarAbs("Toyota");
        // 输出 Beep! Beep!
        myCar.Honk();
        // 输出 Toyota is moving.
        myCar.Move();
    }
}
```

语句解释：

在这个例子中，Vehicle是一个抽象类，它定义了一个普通方法Honk和一个抽象方法Move。CarAbs类继承了Vehicle并实现了Move方法。

抽象类的优缺点（比较）

优点：

- **可以提供默认实现。**抽象类可以包含部分实现，减少子类需要重复实现的代码。在Vehicle示例中，Honk方法是所有Vehicle子类共享的默认行为，子类无需重复实现。
- **允许包含状态和字段。**抽象类可以定义字段和属性，用于**存储和共享**状态。在Vehicle示例中，Name是一个字段，CarAbs子类可以直接继承和使用。适合在复杂的继承层次结构中，提供**一致的**代码结构。

缺点：

- **单继承限制。**一个类只能继承一个抽象类，**限制了继承的灵活性**。Car已经继承了Vehicle，它无法再继承另一个抽象类。
- **不适用于无关类。**抽象类适合用于有**共同逻辑**的类，但对于无关的类（比如Bird和Car），它们不可能从同一个抽象类继承。通常需要设计一个**合理**的继承结构，对于复杂系统可能会引入**不必要的复杂性**。

接口与抽象类的区别（总结）

特点	接口	抽象类
定义	定义类应该实现的方法或属性，但不提供具体实现。	可以包含方法实现和抽象方法（没有实现）。
方法体	所有方法都是抽象的，没有方法体。	可以包含方法体，也可以包含抽象方法。
字段（变量）	不能定义字段。	可以包含字段，并可通过构造函数初始化。
访问修饰符	成员默认是 `public`，不能包含 `private` 或 `protected` 成员。	可以有 `public`、`protected`、`private` 等修饰符。
继承关系	一个类可以实现多个接口（多实现）。	一个类只能继承一个抽象类（单继承）。
包含内容	只能包含方法签名、属性、事件和索引器。	可以包含方法实现、抽象方法、字段、构造函数、属性等。
实例化	接口不能被实例化。	抽象类不能被直接实例化，必须通过子类实现。
用途	用于定义不同类之间共享的行为（更灵活的设计）。	用于定义共享代码，提供某些默认行为。

补充：

- **接口用于定义行为规范，强调“做什么”（What to do）**
- **抽象类用于定义基本功能和扩展，强调“是什么”（What it is）**
- **接口不支持构造函数，抽象类可以。**

如何选择？

1. **使用接口**：
   - 如果需要定义一组行为，而不关心具体实现。
   - 如果需要支持多继承，比如一个类需要实现多个接口。
2. **使用抽象类**：
   - 如果需要定义一个基础类，并提供部分功能的默认实现。
   - 如果需要在基类中定义字段或构造函数。

【Unity基础教程】重点知识汇总

（十二）

Unity C#接口与抽象类