Cha3-认识 Unity3d 的好基友 C# 07

在上一章的内容中，我们一起学习了 C#中的函数。

而在这一课的内容中，我们将学习 C#的另外一个非常重要的概念，那就是类和方法。

让我们开始吧~

个人微信号：iseedo

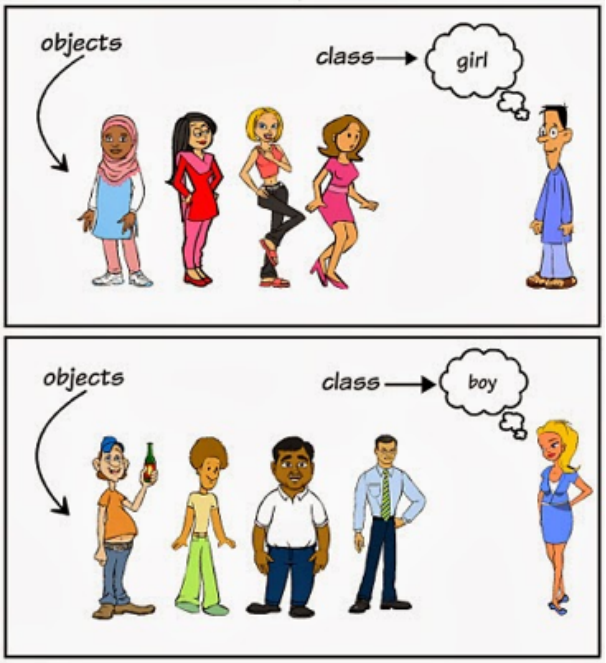
微信公众号：vrlife

07 认识 Unity3d 的好基友 C#-C#中的类和方法

C#是一门面向对象的编程语言，而类、对象和方法则是面向对象中的重要概念。

具有相同属性和功能的一组对象的集合就是一个类。比如人是一个类，猫是一个类。

类的一个实体就是对象。你，我都是“人”这个类的一个对象，如图所示。



至于方法，它的形式和函数类似，也是为了实现某些特定功能的代码块。但是方法和函数的区别在于，方法通常是某个对象所特有的。换而言之，函数是代码世界里独立的生命体，它不依赖于某个对象而生存。但方法不同，任何一个方法都是某个对象的附属生命（寄生体？），同时也只有它的宿主对象才能调用这个方法。

类的继承

继承就是在类之间建立一种“父子”关系，子类可以拥有某些来自父类的属性和方法

父类也被称为基类，子类也被称为派生类。

这么说有点抽象，我们还是用实际的例子来说明。比如在某个幻想类 RPG 游戏中，里面有多种英雄角色，如刺客、战士、法师、术士、弓箭手等等。那么我们可以创建一个英雄角色类，这些英雄角色有一些共同拥有的基本属性，比如昵称、生命值、法术值、法术攻击力、耐力、敏捷等等。这些属性值通常是以变量的形式来保存的。

而某种特定类型的英雄角色则是英雄角色类的子类。

以弓箭手为例，他拥有英雄角色的基本属性，但同时具备自己特有的很多技能，比如后羿射日、元气弹、时空穿梭等等。而这些技能通常是体现在特定的方法中。通过将这些变量和方法放在一起，我们就创造了一个弓箭手的类，这个过程称之为封装。而这个弓箭手的类代表了一个抽象的弓箭手。

以王者荣耀为例，我们可以为所有的英雄创建一个父类，然后每种不同类型的英雄又继承自这个父类，从而形成坦克、战士、刺客、法师、射手和辅助子类。

而在战士这个子类中，又派生出子类程咬金、夏侯惇、达摩、刘备、凯等等具有各自不同特色技

能的战士。

理论充电-Public 和 Private 访问修饰符

在 C#的学习中，我们可能会在不同的场合看到 public 和 private 这两个关键词，又被称之为访问修饰符。

其实除了 public 和 private，还有两个修饰词是 protected 和 internal，但是 public 和 private 用的频率要远远超过另外两个。此外还有一个 protected internal。

需要注意的是，类、属性和方法的前面都可以添加访问修饰符。

1.public

可以从当前程序集（比如同一个项目），或是引用它的其它程序集的任何一段代码中进行访问。

2.private

只有当前类（结构体）才能访问。

3.protected

只有当前类（结构体）或是子类才能访问。

4.interntal

可以从当前程序集的任何一段代码中进行访问，但子类除外。

5.protected internal

可以从当前程序集的任何一段代码中进行访问

visibility / keyword	Containing Classes	Derived Classes	Containing Assembly	Anywhere outside the containing assembly
public	yes	yes	yes	yes
protected internal	yes	yes	yes	no
protected	yes	yes	no	no
private	yes	no	no	no
internal	yes	no	yes	no

比如《王者荣耀》中的英雄角色可以看做一个基类，然后坦克、战士等看做它的子类，而夏侯惇、刘备等则是战士类的子类。

那么一些基本的共有属性（生命值，法力值等）可以设置为 public 或 protected 类型。而每个英雄所特有的技能显然只能是 private 类型的。

好了，说了这么多抽象的知识，还是让我们用一个简单的示例来说话吧。

【示例】在项目中创建并引用一个弓箭手类

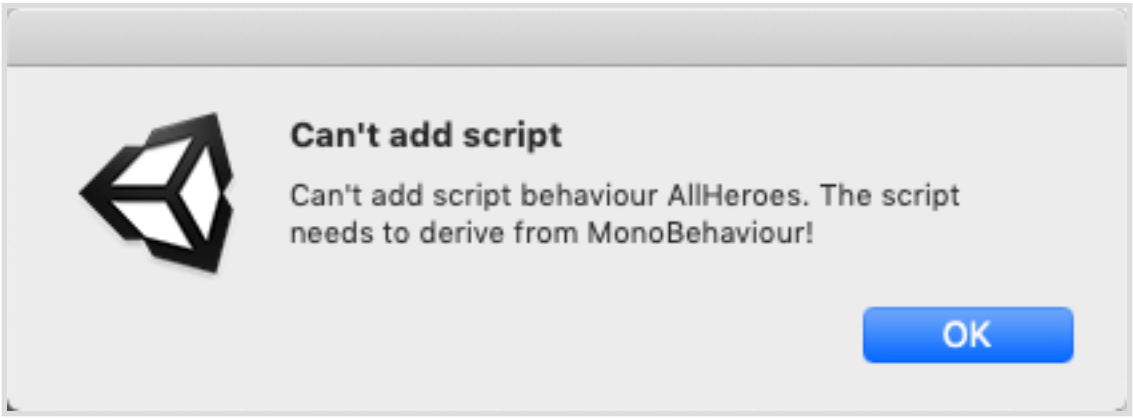
Step1. 首先创建一个新的项目，将其命名为 GloryHeros。

Step2. 在 Project 视图中新建文件夹_Scripts，并创建一个新的脚本文件，名为 AllHeroes。

双击在 Visual Studio 中打开，并输入以下代码：

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

 //1.定义所有英雄的基类
public class AllHeroes
{

    //2.定义所有英雄共同具备的属性



    public int maxLife; //英雄的最大生命值
    protected int maxMagic;//英雄的最大法力值
    protected int heroSpeed; //英雄的移动速度

    protected int currentLife;//英雄的当前生命值
    private int currentMagic;//英雄的当前法力值

    protected bool isHeroDeas;//英雄是否死亡的状态

    //3.定义英雄的复活方法
    public void HeroReborn()
    {

        Debug.Log("英雄从基地复活！");
    }

    //4.定义英雄的死亡方法
    public void HeroDie()
    {
        Debug.Log("英雄 GG 了");
    }

    //5.显示英雄的当前状态
```

```
    protected void ShowHeroStatus()
    {

        Debug.Log("英雄的当前生命值是： " + currentLife);

        Debug.Log("英雄的当前法力值是： " + currentMagic);
```




```
    }
}
```

虽然在上面写了一大堆代码，其实都很容易看懂，我们按照注释行的数字编号来大概解释下：

1.这里定义了所有英雄的基类（父类）。

细心的童鞋可能会发现，跟之前的代码不同，AllHeroes 的后面没有：MonoBehaviour，也就是说它不是继承自 MonoBehaviour，因而不能作为游戏对象的行为组件来影响游戏对象，当然也没办法在 Inspector 视图中关联到某个游戏对象了~

如果你尝试这么操作，就会看到下面的提示：

2.定义了一大堆属性变量，而且用了不同的访问修饰符。

之所以用了不同的访问修饰符，是为了让大家稍后感受不同修饰符的作用~

3.定义了英雄的复活方法

内容很简单，就是在 Console 视图中输出一行调试信息而已。

4.定义了英雄的死亡方法

内容很简单，就是在 Console 视图中输出一行调试信息而已。

5.定义了显示英雄当前状态的方法

内容很简单，就是在 Console 视图中输出两行调试信息而已。

Step3.在 Project 视图中打开文件夹_Scripts，继续创建一个新的脚本文件，名为 MyHero。

双击在 Visual Studio 中打开，并输入以下代码：

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

//1.定义 MyHero 类，继承自 AllHeroes 类


public class MyHero : AllHeroes //MyHero 类继承自 AllHeroes 类
{
    //2.定义我的英雄所特有的属性
    private string heroName;//英雄的名称


    //3.定义 MyHero 所特有的技能
    //4.开启范围攻击
    public void RangeAttack()
    {

        Debug.Log("开始范围攻击");
    }


    //5.开启法术攻击
    private void MagicAttack()
    {
        Debug.Log("开启法术攻击");


    }

}
```

以上代码也很好理解，这里简单解释一下：

1. 创建了一个 MyHero 类，它继承自 AllHeroes 类

2. 定义我的英雄所特有的属性

3. 定义 MyHero 所特有的技能

4. 定义范围攻击的技能

5. 定义法术攻击的技能

Step4. 接下来回到 Unity 编辑器，在 Hierarchy 视图中新建一个游戏对象，命名为 HeroCommandObject。然后在 Inspector 视图中使用 Create And Add 的方式创建一个新的脚本，名为 HeroCommandScript，双击在 Visual Studio 中打开。

输入以下代码：

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


//1.定义 heroCommandScript 类，继承自 MonoBehaviour
public class HeroCommandScript : MonoBehaviour
{
    //2.创建一个 MyHero 类的实例对象
    MyHero myHero = new MyHero();

    // Start is called before the first frame update
    void Start()
    {
        //3.读取并设置英雄的最大生命值
        myHero.maxLife = 30000;

        //4.输出信息
        Debug.Log("英雄的最大生命值是： " + myHero.maxLife);

        //5.调用 HeroReborn 方法
        myHero.HeroReborn();

        //6.调用 RangeAttack 方法
        myHero.RangeAttack();
```

```
        //7.调用 HeroDie 方法
        myHero.HeroDie();

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

虽然很容易看懂，不过还是简单解释下：

1.定义 HeroCommandScript 类，继承自 MonoBehaviour

注意，这行代码不是我们手动添加的，而是在创建脚本时自动添加的。

2.创建一个 MyHero 类的实例对象

注意这行代码的写法，我们通常使用这种方式来创建一个类的实例对象。

ClassName objectName = new ClassName();

3.读取并设置英雄的最大生命值

需要注意的是，maxlife 是我们在 AllHeroes 类中所定义的英雄属性，因为 MyHero 类继承了 AllHeroes 类的属性，所以天然也有这个属性。但是，如果你希望用类似方式来访问其它属性，会出现错误提示。为什么呢？

因为在 AllHeroes 类中，只有 maxLife 是 public 类型的，而其它的都是 protected 或是 private，所以在 HeroCommandScript 这个第三方类的实现代码中当然无法访问了~

4.输出一行特定信息

5,6,7,分别调用了 MyHero 类的三个方法

需要注意的是，HeroReborn 和 HeroDie 方法都是 AllHeroes 类中所定义的 public 类型方法。而 RangeAttack 则是在 MyHero 类中所定义的 public 类型方法。

至于添加了 protected 和 private 访问修饰符的方法，我们是无法在这里调用的~

好了，点击工具栏上的播放按钮，就可以在 Console 面板中看到对应的输出结果。

最后再次强调，在 Unity 中有一些非常重要的类，而其中最为重要的就是 MonoBehaviour，它是所有**游戏**对象脚本组件的基类。关于 Unity 中这些重要类的详细信息，请参考 Unity3d 的官方文档。

当然，我们在后续的内容中也会逐渐涉及到的~

好了，关于 C#中的类、对象和方法，我们就先介绍到这里。

在下一课的内容中，我们将一起接触编程开发中极其重要极其有用的一个理论基础。

至于是什么？让我们下一课再见~