# Cha3-认识 Unity3d 的好基友 C# 04

在上一章的内容中，我们了解了 C#中的变量和基本数据类型。

在这一课的内容中，我们将进一步了解 C#的表达式与运算符。

还等什么呢？让我们开始吧~

个人微信号：iseedo

微信公众号：vrlife

**04 认识 Unity3d 的好基友 C#-C#中的运算符和表达式**

对于几乎所有的童鞋来说，对数学中的运算符和表达式都不会陌生。毕竟从小学开始，我们就已经接触了最基本的四则运算。

在计算机中，表达式与运算符的作用是对数据或信息进行各种形式的运算处理，构成了程序代码的主体。

表达式由运算符和操作数组成。其中运算符比较好理解，它用于设置对操作数进行怎样的运算，例如+，-，X 和/分别代表加减乘除四种运算。而操作数是运算符作用于的实体，指出指令执行的操作所需要的数据来源。操作数的概念最早来源于汇编语言，其代表参与运算的数据及其单元地址。在 C#中，操作数可以简单理解为参与运算的各类变量和表达式等。

在 C#中，我们需要了解以下几种主要的运算符。

## 算术运算符

算术运算符是我们都很熟悉的基本数学运算，主要是加减乘除求余数，下表显示了 C#语言所支持的所有算术运算符。

假设变量 A 的值是 10，变量 B 的值是 20，那么：

| 运算符 | 描述 | 实例 |
| --- | --- | --- |
| + | 把两个操作数相加 | A + B 将得到 30 |
| - | 从第一个操作数中减去第二个操作数 | A - B 将得到 -10 |
| * | 把两个操作数相乘 | A * B 将得到 200 |
| / | 分子除以分母 | B / A 将得到 2 |
| % | 取模运算符，整除后的余数 | B % A 将得到 0 |
| ++ | 自增运算符，整数值增加 1 | A++ 将得到 11 |
| -- | 自减运算符，整数值减少 1 | A-- 将得到 9 |

这部分的理论知识没有太多好讲的，毕竟我们都曾经上过小学~

接下来还是通过实际的例子来演示一下。

【示例练习】实现两个数字之间的基本数学运算。

打开 Unity，新创建一个项目，将其命名为 BasicMath。在 Project 视图中右键单击空白处，选择 Create-Folder，将其命名为_Scripts。双击进入该文件夹，然后单击鼠标右键，选择 Create→C# Script。以上这些操作就不再重复了。

把新创建的脚本文件更名为 OperationScript，然后双击在 Visual Studio 中打开，并更改其中的代码如下：

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

public class OperationScript : MonoBehaviour
{
    // Start is called before the first frame update

    void Start()
```

```
    {
        //1.这是一行注释，解释了下面代码的作用，也即定义两个整形变量

        int firstNumber = 1;    //第 1 个数
        int secondNumber = 5;   //第 2 个数

        //2.求二者的和
        int sumOfNumbers = firstNumber + secondNumber;

        //3.求二者的积
        int mulOfNumbers = firstNumber * secondNumber;

        //4.求二者的差
        int difOfNumbers = firstNumber - secondNumber;

        //5.求二者的商并进行强制类型转换
        double divOfNumbers = (double)firstNumber /
secondNumber;

        //6.求二者的余数
        int remOfNumbers = firstNumber % secondNumber;

        //7.输出以上计算结果到 Console 中

        Debug.Log("两个数字的和是: " + sumOfNumbers);
        Debug.Log("两个数字的乘积是: " + mulOfNumbers);
        Debug.Log("两个数字之间的差是: " + difOfNumbers);
        Debug.Log("两个数字相除的商是: " + divOfNumbers);
        Debug.Log("两个数字相除的余数是: " + remOfNumbers);

    }
```

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

    }
}
```

接下来**还是按照注释中的数字编号顺序来详细解释一下每行代码的作用。**

1. 接下来的两行代码分别定义了两个整数变量 firstNumber 和 secondNumber，它们的值分别是 1 和 5。
2. 这行代码定义了一个名为 sumOfNumbers 的整数变量，用它来保存两个整数的和
3. 这行代码定义了一个名为 mulOfNumbers 的整数变量，用它来保存两个整数的乘积
4. 这行代码定义了一个名为 difOfNumbers 的整数变量，用它来保存两个整数的差值
5. 这行代码定义了一个名为 divOfNumbers 的浮点型变量，用它来保存两个整数相除的商，并将运算结果强制转换成浮点数。
6. 这行代码定义了一个名为 remOfNumbers 的整数变量，用它来保存两个整数相除的余数
7. 接下来用 Debug.Log 函数分别输出了以上的 5 种数学运算的结果。

下 Ctrl+S 保存代码修改，然后回到 Unity 编辑器主界面。

猜猜看接下来我们要做什么？

当然是要把这个脚本文件和某个游戏对象关联在一起了~

这次我们会尝试用另外一种方法创建游戏对象，同时换用一种方法将脚本和游戏对象关联在一起，作用和之前的一样。

从 Unity 的顶部菜单中选择 Game Object-Create Empty，可以看到 Hierarchy 视图中出现了一个名为 GameObject 的游戏对象。这个操作的具体含义就是添加了一个空白游戏对象。然后右键单击

GameObject，选择 Rename，把它更名为 OperationScriptObject。

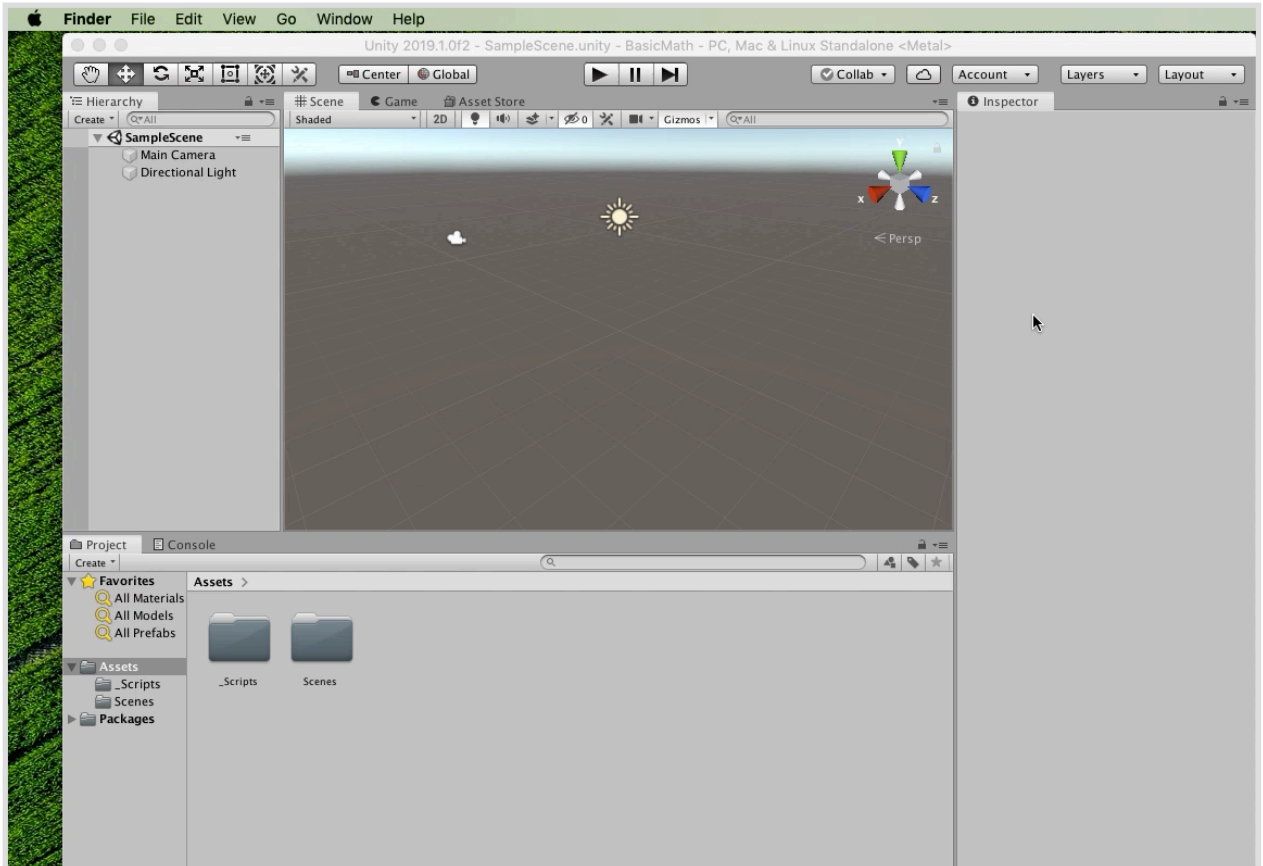
接下来的操作需要一点小技巧~

选中 OperationScriptObject 这个游戏对象，点击 Inspector 视图中最右上角的有个小锁一样的标记，表示在这个视图中锁定当前选择。

然后在 Project 视图中找到 OperationScript 这个脚本文件，按下鼠标左键不放，把它拖动到 Inspector 视图的空白区域，会看到 OperationScript 成了 OperationScriptObject 这个空白游戏对象的行为脚本组件。

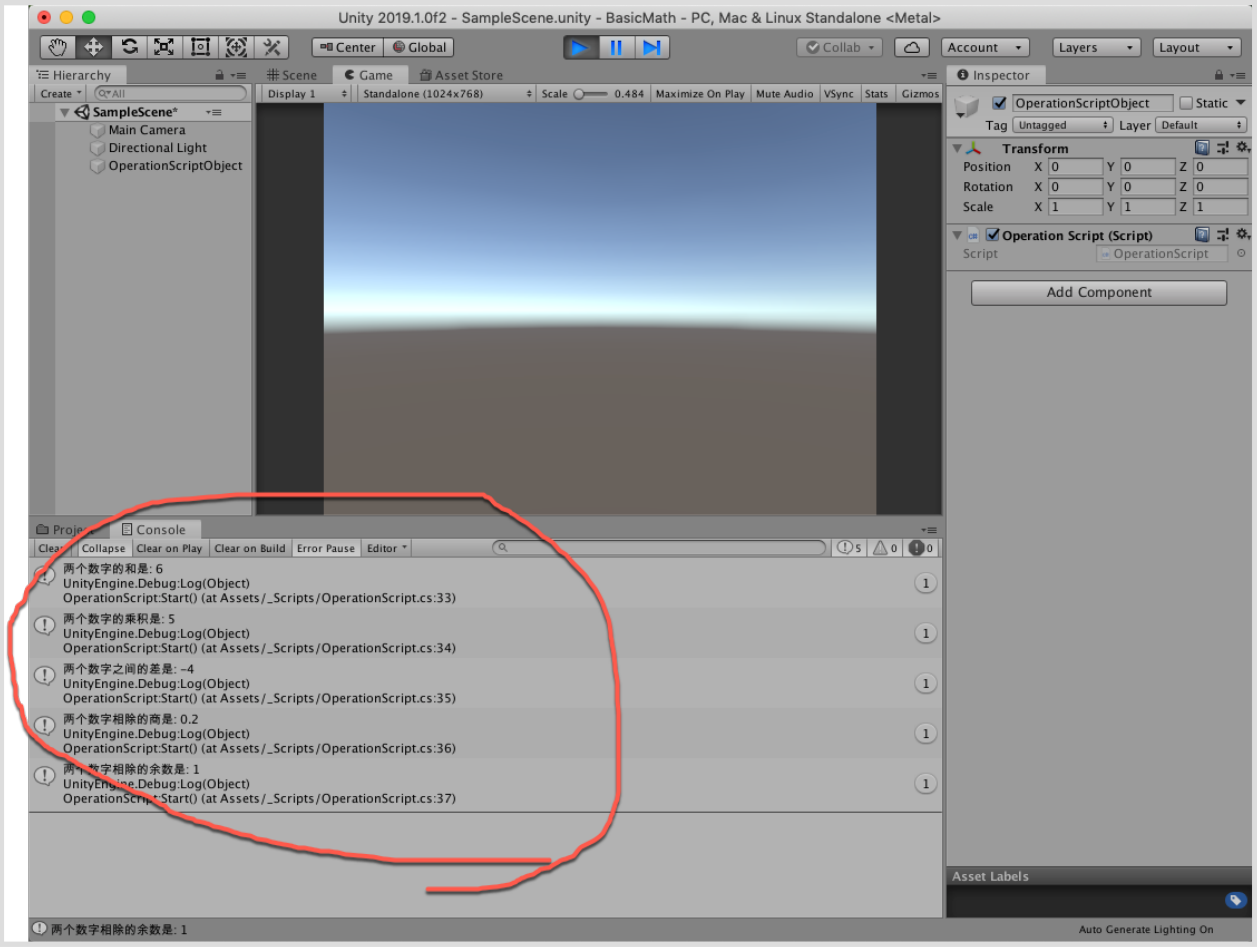
具体的操作请参考下面的微视频~

再次强调，Script 脚本的文件名和 visual studio 中的类名必须完全一致，否则这个拖动的操作是无法完成的。如果因为误操作使得脚本文件名和类名不一致，就需要手动在代码中更改类名保持一致。



单击 Unity 编辑器工具栏上的播放控制按钮，即可在 Console 里面看到输出的结果。

### 赋值运算符




在 C#中，赋值运算符用于将一个数据赋予一根变量、属性或者引用。数据本身是常量、变量或者表达式。赋值运算符本身又分为简单赋值和复合赋值，其作用如下表所示：

| 赋值运算符 | 表达式示例 | 含义 |
| --- | --- | --- |
| = | x = 10 | 将10赋给变量x。 |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| >>= | x >>= y | x = x >>= y |
| <<= | x <<= y | x = x <<= y |
| &= | x &= y | x = x & y |
| \|= | x \|= y | x = x \| y |
| ^= | x ^= y | x = x ^ y |

【示例】测试使用赋值运算符。

回到 BasicMath 项目，在 Project 视图中的_Scripts 子目录下创建一个新的 C# script，将其命名为 AssignmentScript。

双击在 Visual Studio 中打开该脚本文件，并输入代码如下：

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

public class AssignmentScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        //1.定义一个常量
        const int basicNumber = 10;
        //2.定义一个普通的整数变量
        int x = basicNumber;


        //3.输出不同赋值计算的结果
        Debug.Log("x=" + x);
        Debug.Log("x+=2 的运算结果为：" + (x += 2));
```

```
        x = basicNumber;
        Debug.Log("x-=2 的运算结果为：" + (x -= 2));

        x = basicNumber;
        Debug.Log("x*=2 的运算结果为：" + (x *= 2));

        x = basicNumber;
        Debug.Log("x/=2 的运算结果为：" + (x /= 2));

        x = basicNumber;
        Debug.Log("x%=2 的运算结果为：" + (x %= 2));

        x = basicNumber;
        Debug.Log("x >>= 2 的运算结 果为：" + (x >>= 2));



        x = basicNumber;
        Debug.Log("x<<=2 的运算结果为：" + (x <<= 2));

        x = basicNumber;
        Debug.Log("x&=2 的运算结果为：" + (x &= 2));

        x = basicNumber;
        Debug.Log("x|=2 的运算结果为：" + (x |= 2));

        x = basicNumber;
        Debug.Log("x^=2 的运算结果为：" + (x ^= 2));

    }

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

```
    }
}
```

**需要注意的是，我们仅仅修改了 void Start(){}里面的代码。**

另外对于初学者需要特别注意的是，以上代码中所用到的双引号和分号必须采用英文的半角输入，如果使用全角输入，是没办法得到想要结果的。**这个也是新手常犯的错误之一~**

**小练习：**

1. 尝试着给以上代码添加完整的注释。

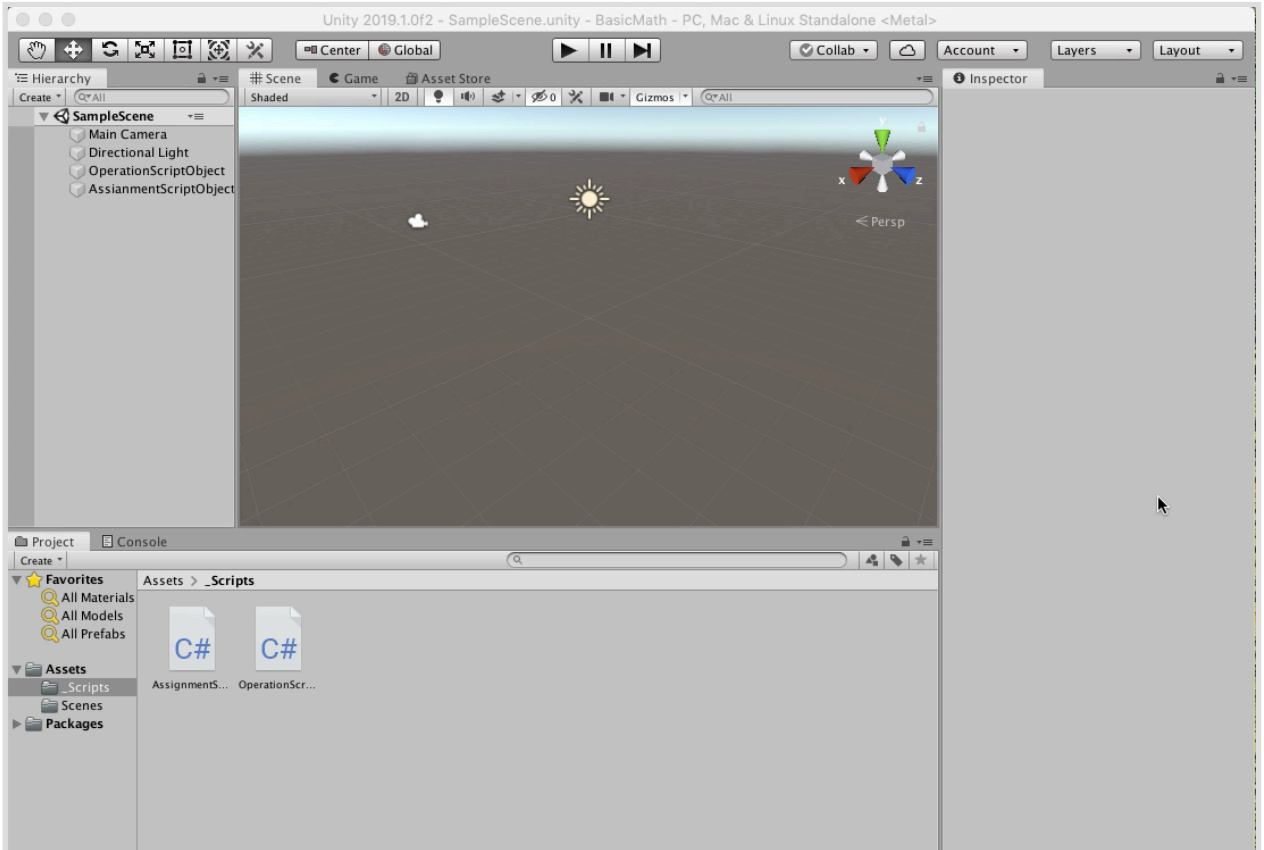
2. 在场景中添加一个新的物体，命名为 AssignmentScriptObject,然后把 AssignmentScript 脚本和它关联在一起。需要提醒大家的是，因为刚才的操作中锁定了 Inspector 视图中的内容，所以在关联新的脚本时，必须点击 Inspector 视图上的小锁标记取消锁定~

好了，关于上面的这个小练习，希望大家自行完成。接下来的内容是假定大家已经搞定了上面的两步了~

如果有问题，请通过论坛提问，或者跟童鞋一起商量。(http://icode.fun)

为了不影响查看赋值运算的结果，我们需要暂时屏蔽刚才的四则运算结果。

怎么破？一个很简单的方式，在 Hierarchy 视图中选中 OperationScriptObject，然后在 Inspector



视图中取消顶部的勾选框，或是取消 Operation Script(Script)组件的勾选框。这两种操作的区别是，如果取消顶部的勾选框，那么就意味着 OperationScriptObject 这个游戏对象就不会出现在场景中。而如果只是取消 Operation Script(Script)组件的勾选框，就意味着这个脚本不会发挥作用控制游戏对象的行为，但是游戏对象本身还是会出现在场景中~

此外，需要在测试运行之前手动切换到 Console 视图，点击 Clear 按钮以清除之前的显示内容。

最后点击 Unity 编辑器上的播放按钮，切换到 Console 视图，就可以看到我们想要的结果了。

具体的操作可以参考下面的微视频。

## 关系运算符

关系运算符用于比较两个值之间的关系，并在比较之后返回一个布尔类型的运算结果。

常用的关系运算符如下表所示：

| 关系运算符 | 作用说明 |
| --- | --- |
| == | 等于 |
| < | 小于 |
| <= | 小于或等于 |
| > | 大于 |
| >= | 大于或等于 |
| != | 不等于 |

**相信大家对理解关系运算符本身没什么问题，还是看看具体怎么用吧~**

【示例】使用关系运算符。

回到 BasicMath 项目，在 Project 视图中的_Scripts 子目录下创建一个新的 C# Script，命名为 ComparisionScript。

双击在 Visual Studio 中打开该脚本文件，并输入代码如下：

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

```
public class ComparisionScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        int firstNumber = 3;
        int secondNumber = 5;

        if (firstNumber == secondNumber)
        {
            Debug.Log("两个数字相同");
        }
        if (firstNumber < secondNumber)
        {
            Debug.Log("第一个数字比第二个数字小。");

        }
        if (firstNumber <= secondNumber)
        {
            Debug.Log("第一个数字小于或等于第二个数字。");
        }
        if (firstNumber > secondNumber)
        {
            Debug.Log("第一个数字比第二个数字大。");
        }
        if (firstNumber >= secondNumber)
        {
            Debug.Log("第一个数字大于或等于第二个数字。");
        }
        if (firstNumber != secondNumber)
        {
            Debug.Log("第一个数字不等于第二个数字。");

        }

    }

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

```
    {

    }
}
```

以上代码比较简单，我们使用了逻辑判断，比较 firstNumber 和 secondNumber 的大小，并根据比较的结果来输出不同的结果。

**小练习：**

1. 给以上代码添加完整的注释，解释其作用.

2. 猜猜看输出的结果会是怎样的。

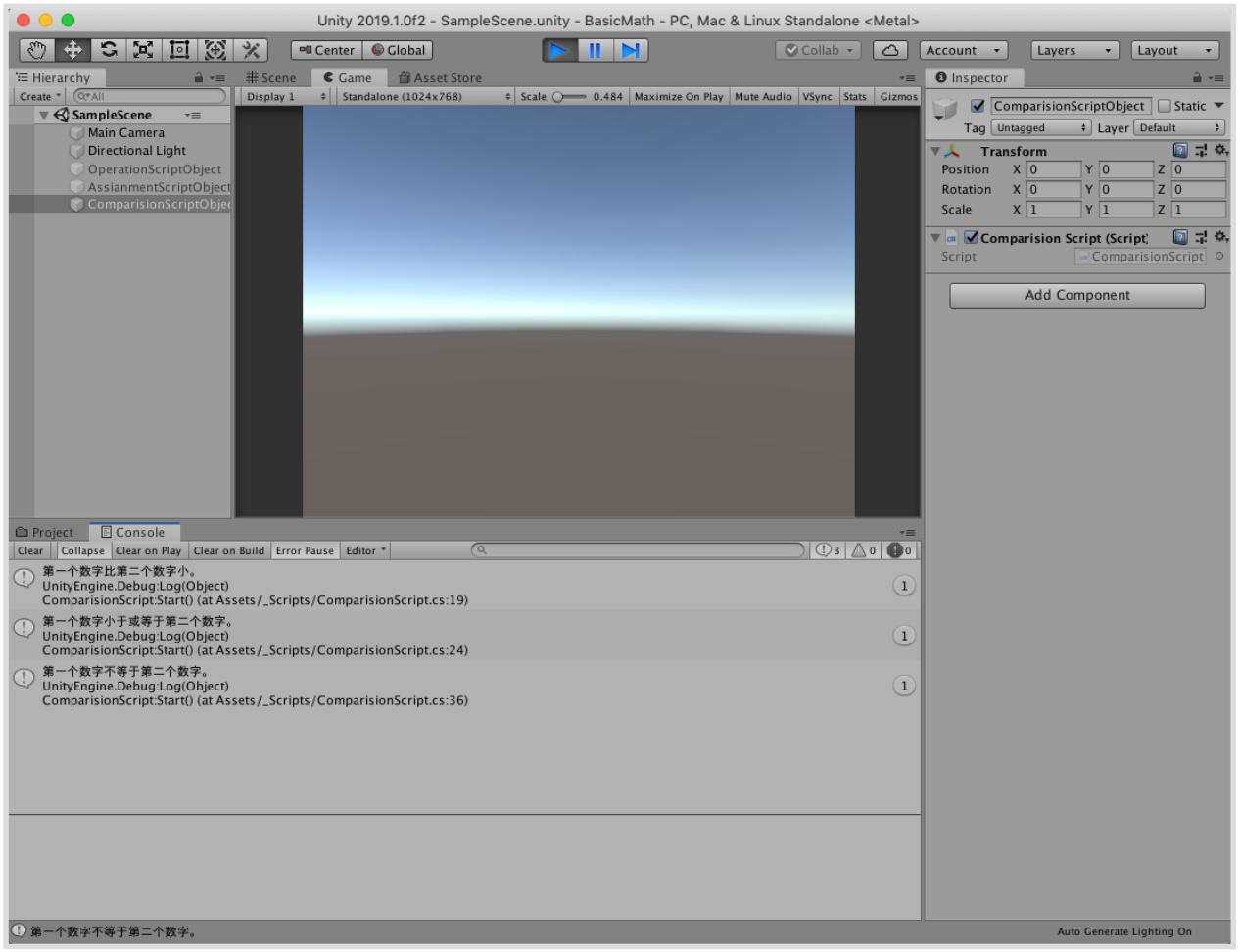
3. 在场景中添加一个新的空白游戏对象，并将刚才所新创建的游戏脚本关联到这个游戏对象上。类似刚才的操作，禁用 AssignmentScript 脚本组件。

注意：

当我们对场景中的游戏对象做了任何修改之后，需要从菜单中选择 File-Save，或者用快捷键 Ctrl+S(Mac 下是 Command +S)，对修改进行保存。

好了，当你成功的完成了以上练习之后，就可以愉快的点击 Unity 编辑器上的 Play 按钮，然后在 Console 视图中查看结果了。

## 条件运算符




条件运算符用于进行逻辑判断，并返回一个布尔类型的运算结果。**我们的布尔大大终于要出场了，激动~**

常用的条件运算符如下表所示。

| 条件运算符 | 作用说明 |
| --- | --- |
| && | 与 |
| ‖ | 或 |
| ! | 非 |
| ?: | 三目运算符 |

关于条件运算，这里要多说两句，毕竟这个东东不光小学没有学，目测中学好像也没有用过。

因为布尔数学属于计算机科学的基础之一。对于计算机科班出身的我，记忆犹新的是在本科上过一门课叫《离散数学》。或许跟微积分概率统计数值分析这些学科杀手比起来稍好一点，但离散数学这本课绝对也是让人头疼的一门课。

很多人都知道对于计算机学科来说，算法和数据结构是基础的基础。然而很多非计算机专业的童鞋不知道，其实离散数学可以说的上是算法的基础~

当然，对于非计算机专业的小白来说，Don’t panic。不要恐慌，我们这里要接触到的条件运算符只是基础的基础，一点都不难。

实际上最常用的也就是以上的四个。

1.&& 与

代表需要同时满足两个条件。

比如：在《绝地求生》游戏的单人模式下，想要愉快的吃鸡就必须同时满足两个条件：（1）自己还活着 （2）敌人全部 GG

2.|| 或

代表两个条件中至少满足一个

还是拿《绝地求生》为例，在联机模式下，要想吃鸡就要满足以下条件中的至少一个：（1）自己还活着 （2）同组的队友至少还有一个活着

3.！非

代表反过来

比如我们定义了一个变量是 willTommorrowRain，明天是否下雨。

默认状况下变量的值是 true，那么!willTommorrowRain 就是 false。

4.？：

三目运算符

这个相对复杂一点，跟逻辑判断语句有关表达式

a?b:c 其实可以理解成：

如果 a 是 true，那么结果就是 b，否则结果就是 c。

理论上看起来有点抽象，我们还是用实例来说话比较简单~

【示例】使用条件运算符。

回到 BasicMath 项目，在 Project 视图中的_Scripts 子目录下创建一个新的 C# script，命名为 ConditionalScript。双击在编辑器中打开该脚本文件，并输入代码如下：

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

public class ConditionalScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
         //1.定义了两个布尔变量
       bool isPlayer1Dead = true;
       bool isPlayer2Dead = false;



        //2.定义了三个整数变量
        int player1KilledLives = 1;
        int player1InitialLives = 5;
        int player1CurrentLives;


      //3.如果两个玩家角色都已 GG（Game Over，通常指游戏角色死亡）

        if (isPlayer1Dead && isPlayer2Dead) {
          Debug.Log ("所有玩家角色均已 GG");
      }
```

```
        //4.如果至少有一个玩家角色已经 GG

        if (isPlayer1Dead || isPlayer2Dead) {
            Debug.Log ("至少有一个玩家角色已 GG");
          }

        //5.如果第一个玩家角色还没有 GG
         if(!isPlayer1Dead == true){
              Debug.Log ("第一个玩家角色还没有 GG");
          }

        //6.如果第一个玩家角色已经 GG，那么他当前的生命值等于初始生命值减 1，
如果还没有 GG，那么他当前的生命值等于初始生命值

          player1CurrentLives = isPlayer1Dead ?
(player1InitialLives - player1KilledLives) :
player1InitialLives;
          Debug.Log ("第一个玩家还剩下的重生机会是：" +
player1CurrentLives);

    }

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

    }
}
```

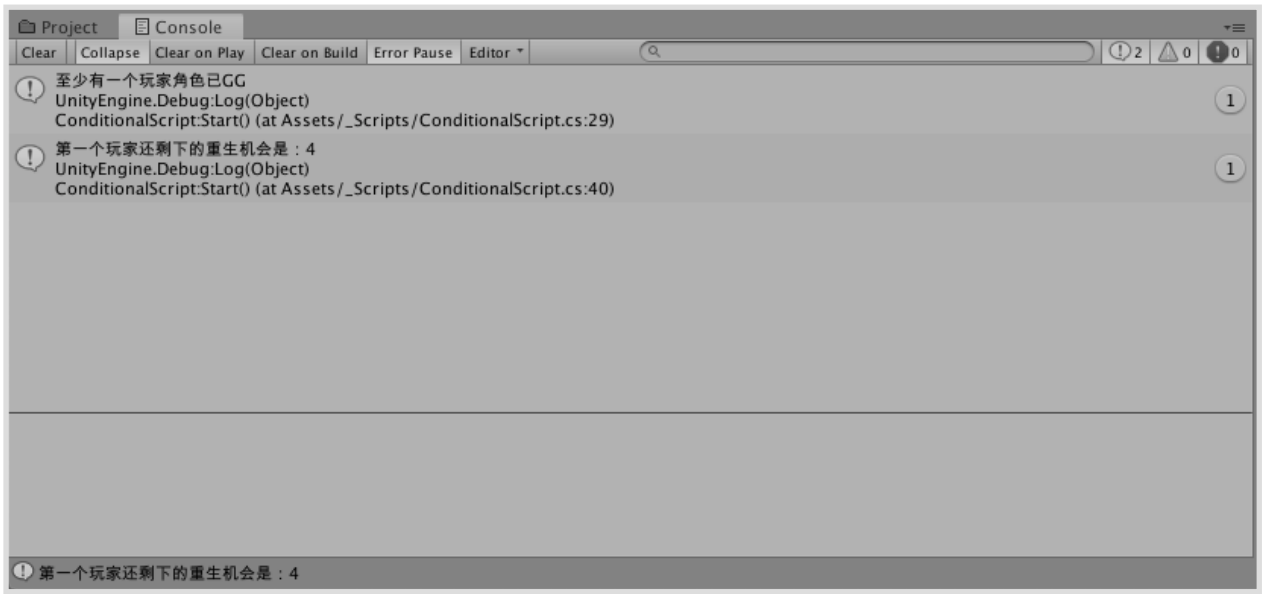
老规矩，还是按照注释行的数字编号来解释下代码的作用：
1.第一行注释下面的两行代码定义了两个布尔类型的变量，isPlayer1Dead 和 isPlayer2Dead，代表玩家 1 是否已死亡，玩家 2 是否已死亡。
2.编号 2 的注释下面定义了三个整数变量，分别代表玩家受伤害后减少的生命值，玩家的初始生命值，和玩家的当前生命值。
3.编号 3 的注释下面是一个逻辑判断语句，用到了&&与
当两个玩家都 GG 了的时候，输出对应的游戏结果。
4.编号 4 的注释下面是一个逻辑判断语句，用到了||或
当两个玩家中的一个 GG 的时候，输出对应的游戏结果。
5.编号 5 的注释下面是一个逻辑判断语句，用到了 !非

当第一个玩家角色还没 GG 的时候，输出对应的游戏结果。
6.编号 6 的注释下面是三目运算符，如果第一个玩家角色已经 GG，那么他当前的生命值等于初始生命值减 1，如果还没有 GG，那么他当前的生命值等于初始生命值
盎然然后输出玩家还剩下的生命值。

**小练习：**

和之前的操作类似，在场景中创建一个新的游戏对象，命名为 ConditionalScriptObject，将 ConditionalScript 游戏脚本和这个新创建的游戏对象关联起来。注意取消对其它几个游戏对象中脚本组件的勾选。

最后点击 Play 按钮，在 Console 视图中查看结果。




好了，通过刚才的代码我们已经知道，在权力的游戏第 8 季中至少有一个主要玩家（比如龙妈，囧.snow.know nothing，珊莎等）已经 GG 领便当了。当然，幸运的是，还有一个最主要的玩家即便挂了，也还有机会再次重生~难道是 snow.know nothing?

欲知何事如何，且听下回分解~

让我们下一课见~