Unity Script

一. GameObject

1. 属性（Properties）

1) activeInHierarchy
定义场景中的游戏对象是否处于可见状态。

2) isStatic
可以获取和设置游戏对象的静态编辑标志。

3) layer
游戏对象所在的层级。

4) Scene
游戏对象所属的场景。

5) sceneCullingMask
用于确定在哪个场景中渲染游戏对象。

6) Tag
这个游戏对象的标签。

7) transform
游戏对象的 transform component。

2. 构造函数（Constructors）

1) GameObject
创建一个名为 name 的新的游戏对象。

3. 公共方法

1) AddComponent
将一个新的组件类添加到游戏对象。

2) BroadcastMessage
在此游戏对象或其任何子项中的每个 MonoBehaviour 上调用名为 methodName 的方法。

3) CompareTag
比较游戏对象的 Tag

4) GetComponent
获取游戏对象上的组件，如果有要获取的组件就返回该类型的组件，如果没有，就返回 null。

5) GetComponentInChildren
使用深度优先搜索返回该游戏对象或其任何子项中该类型的组件。（没有返回空）

6) GetComponentInParent
获取在该游戏对象或其任何父项中指定类型的组件。（有只返回一个，没有返回空）

7) GetComponents
返回该游戏对象中指定类型的所有组件。

8) GetComponentsInChildren

返回该游戏对象或其任何子项中指定类型的所有组件。

9) GetComponentsInParent

返回该游戏对象或其任何父项中指定类型的所有组件。

10) SendMessage

在此游戏对象中的每个 MonoBehaviour 上调用名为 methodName 的方法。

11) SendMessageUpwards

在此游戏对象中的每个 MonoBehaviour 和行为的每个父节点上调用名为 methodName 的方法。

12) SetActive

根据给定的布尔值决定激活还是停用该游戏对象。

13) TryGetComponent

尝试获取指定类型的组件。

4. 静态方法

1) Find

按名称在场景中查找游戏对象并返回它。

2) FindGameObjectsWithTag

根据 Tag 返回一组激活状态的游戏对象。如果未找到游戏对象，则返回空数组。

3) FindWithTag

根据 Tag 返回一个激活状态的游戏对象。如果未找到游戏对象，则返回 null。

二. Transform

1. 公共方法

1) LookAt

① 声明

public void LookAt(Transform target);

public void LookAt(Transform target, Vector3 worldUp = Vector3.up);

② Parameters-参数

target-要指向的对象。

worldUp-指定向上方向的向量。

③ 描述

旋转变换，使前向矢量指向 target 的当前位置。然后它旋转变换以将其向上方向向量指向向量暗示的方向 worldUp。如果省略该 worldUp 参数，该函数将使用世界 y 轴。worldUp 如果向前方向垂直于，则旋转的向上矢量将仅与矢量匹配 worldUp。

④ 声明

public void LookAt (Vector3 worldPosition);

public void LookAt (Vector3 worldPosition , Vector3 worldUp = Vector3.up);

⑤ Parameters-参数

worldPosition-看向某点

worldUp-指定向上方向的向量。

① 描述

旋转变换，使前向向量指向 worldPosition。然后它旋转变换以将其向上方向向量指向向

量暗示的方向 worldUp。如果省略该 worldUp 参数，该函数将使用世界 y 轴。worldUp 如果向前方向垂直于，则旋转的向上矢量将仅与矢量匹配 worldUp。

三. Vector3

1. 描述

3D 向量和点的表示。

这种结构在整个 Unity 中用于传递 3D 位置和方向。它还包含用于执行常见向量运算的函数。

除了下面列出的函数，其他类也可用于操作向量和点。例如 Quaternion 和 Matrix4x4 类对于旋转或变换向量和点很有用。

2. Static Properity-静态属性

1) back

Vector3(0, 0, -1) 的简写。

2) down

Vector3(0, -1, 0) 的简写。

3) forward

Vector3(0, 0, 1) 的简写。

4) left

Vector3(-1, 0, 0) 的简写。

5) negativeInfinity

Vector3(float.NegativeInfinity, float.NegativeInfinity, float.NegativeInfinity) 的简写。

6) one

Vector3(1, 1, 1) 的简写。

7) positiveInfinity

Vector3(float.PositiveInfinity, float.PositiveInfinity, float.PositiveInfinity) 的简写。

8) right

Vector3(1, 0, 0) 的简写。

9) up

Vector3(0, 1, 0) 的简写。

10) zero-零向量

Vector3(0, 0, 0) 的简写。

3. Static Methods-静态方法

1) Angle

计算两个 3 维向量之间的角度

2) ClampMagnitude-限制模

返回一个向量的副本，其大小被限制为 maxLength。

3) Cross

两个向量的叉乘。

4) Distance

返回 a 向量和 b 向量之间的距离。

5) Dot
两个向量的点乘。

6) Lerp
在两点之间线性插值。

7) LerpUnclamped
在两个向量之间线性插值。

8) Max
返回由两个向量的最大分量组成的向量。

9) Min
返回由两个向量的最小分量组成的向量。

10) MoveTowards
计算当前位置和指定的位置之间的位置，移动不超过 maxDistanceDelta 指定的距离。

11) Normalize
使这个向量的大小为 1。

12) OrthoNormalize
使向量标准化并相互正交。（即两两向量内积为 0）

13) Project
将一个向量投影到另一个向量上。

14) ProjectOnPlane
将向量投影到由垂直于该平面的法线定义的平面上。

15) Reflect
从法线定义的平面反射向量。

16) RotateTowards
向目标旋转。

17) Scale
将两个向量相乘。

18) SignedAngle
计算向量 from 和 to 之间相对于轴的有符号角。

19) Slerp
在两个向量之间进行球面插值。

20) SlerpUnclamped
在两个向量之间进行球面插值。

21) SmoothDamp
随着时间的推移，逐渐将向量改变为期望的目标。

4. 向量归一化的区别

1) 不同点

Vector3.normalized：不改变当前向量，返回一个新的规范化的向量。
Vector3.Normalized：改变当前向量，当前向量长度为一。

2) 相同点

实现规范化，让一个向量保持相同的方向，但它的长度为 1，如果这个向量太小而不能被规范化，一个零向量会被返回。

四. KeyCode 编码

值	描述	值	描述
BackSpace	退格键	Delete	正向删除键
Tab	制表键	Clear	清除键
Return	回车键	Pause	PC 上的暂停键
Escape	退出键	Space	空格键
Keypad0	数字小键盘 0	Keypad1	数字小键盘 1
Keypad2	数字小键盘 2	Keypad3	数字小键盘 3
Keypad4	数字小键盘 4	Keypad5	数字小键盘 5
Keypad6	数字小键盘 6	Keypad7	数字小键盘 7
Keypad8	数字小键盘 8	Keypad9	数字小键盘 9
KeypadPeriod	数字小键盘句号	KeypadDivide	数字小键盘除号
KeypadMultiply	数字小键盘乘号	KeypadMinus	数字小键盘减号
KeypadPlus	数字小键盘加号	KeypadEnter	数字小键盘回车键
KeypadEquals	数字小键盘等号	UpArrow	上箭头键
DownArrow	下箭头键	RightArrow	右箭头键
LeftArrow	左箭头键	Insert	插入键
Home	起始键	End	结束键
PageUp	上一页	PageDown	下一页
F1	F1 功能键	F2	F2 功能键
F3	F3 功能键	F4	F4 功能键
F5	F5 功能键	F6	F6 功能键
F7	F7 功能键	F8	F8 功能键
F9	F9 功能键	F10	F10 功能键
F11	F11 功能键	F12	F12 功能键
Alpha0	键盘顶部的数字键 0	Alpha1	键盘顶部的数字键 1
Alpha2	键盘顶部的数字键 2	Alpha3	键盘顶部的数字键 3
Alpha4	键盘顶部的数字键 4	Alpha5	键盘顶部的数字键 5
Alpha6	键盘顶部的数字键 6	Alpha7	键盘顶部的数字键 7
Alpha8	键盘顶部的数字键 8	Alpha9	键盘顶部的数字键 9
Exclaim	感叹号键	DoubleQuote	双引号键
Hash	哈希键	Dollar	美元符号键
Percent	百分号键	Ampersand	&键
Quote	单引号键	LeftParen	左括号键
RightParen	右括号键	Asterisk	星号键
Plus	加号键	Comma	逗号键
Minus	减号键	Period	句号键
Slash	斜杠键	Colon	冒号键
Semicolon	分号键	Less	小于符号键

Equals	等号键	Greater	大于符号键
Question	问号键	At	@符号键
LeftBracket	左中括号键	Backslash	反斜杠键
RightBracket	右中括号键	Caret	托字符键
Underscore	下划线键	BackQuote	反引号键
A	字母 A 键	B	字母 B 键
C	字母 C 键	D	字母 D 键
E	字母 E 键	F	字母 F 键
G	字母 G 键	H	字母 H 键
I	字母 I 键	J	字母 J 键
K	字母 K 键	L	字母 L 键
M	字母 M 键	N	字母 N 键
O	字母 O 键	P	字母 P 键
Q	字母 Q 键	R	字母 R 键
S	字母 S 键	T	字母 T 键
U	字母 U 键	V	字母 V 键
W	字母 W 键	X	字母 X 键
Y	字母 Y 键	Z	字母 Z 键
LeftCurlyBracket	左大括号键	Pipe	|
RightCurlyBracket	由大括号键	Tilde	波浪号
Numlock	小键盘锁定键	CapsLock	大写锁定键
ScrollLock	滚动锁定键	RightShift	右换挡键
LeftShift	左换挡键	RightControl	右控制键
LeftControl	左控制键	RightAlt	右 Alt 键
LeftAlt	左 Alt 键	LeftMeta	如果在输入管理器设置中启用了物理键，则映射到左 Windows 键或左命令键，否则仅映射到左命令键。
LeftCommand	左命令键	LeftApple	左 Apple 键
LeftWindows	左 Windows 键	RightMeta	如果在输入管理器设置中启用了物理键，则映射到右侧 Windows 键或右侧 Command 键，否则仅映射到右侧 Command 键。
RightCommand	右命令键	RightApple	右 Apple 键
RightWindows	右 Windows 键	AltGr	右侧更改键
Help	帮助键	Print	打印键
SysReq	系统请求键	Break	中断键
Menu	菜单键	Mouse0	第 1 个鼠标键（鼠标

			左键）
Mouse1	第 2 个鼠标键（鼠标右键）	Mouse2	第 3 个鼠标键（鼠标中键）
Mouse3	第 4 个鼠标键	Mouse4	第 5 个鼠标键
Mouse5	第 6 个鼠标键	Mouse6	第 7 个鼠标键

五. 数学函数库

1. 绝对值-Mathf.Abs

① public static float Abs(float f);

返回浮点型的绝对值

② public static int Abs(int value);

返回整型的绝对值

2. 反余弦-Mathf.Acos

public static float Acos(float f);

根据余弦值，返回弧度。

3. 近似-Mathf.Approximately

public static bool Approximately(float a, float b);

比较两个浮点数值，看它们是否非常接近，由于浮点数值不精确，不建议使用等于来比较它们。

4. 反正弦-Mathf.Asin

public static float Asin(float f);

根据正弦值，返回弧度。

5. 反正切-Mathf.Atan2

public static float Atan2(float x, float y);

根据 Y 除以 X，返回对应的弧度。返回值表示相对直角三角形，对角的角，其中 X 是临边边长，Y 是对边边长。返回值是 X 轴和一个二维向量开始于 0，结束在(x,y)处之间的角。

6. 反正切-Mathf.Atan

public static float Atan(float f);

根据正切值，返回以弧度为单位的角度，返回值介于负二分之π和正二分之π之间。

7. 向上最小整数- Mathf.CeilToInt

public static int CeilToInt(float f);

返回大于或等于 f 的最小整数，返回值是整型。

8. 上限值-Mathf.Ceil

public static float Ceil(float f);

返回大于或等于 f 的最小整数，返回值是浮点型。

9. 范围限制 0-1-Mathf.Clamp01

public static float Clamp01(float value);

限制值的范围在 0 到 1 之间。如果值小于 0 则返回 0，值大于 1 则返回 1。

10. 限制- Mathf.Clamp

① public static float Clamp(float value, float min, float max);

如果给定的浮点值小于最小值，则返回最小值。如果给定值大于最大值，则返回最大值。

使用 Clamp 将值限制在由最小值和最大值定义的范围内。

注意：如果最小值大于最大值，则该方法返回最小值。

② public static int Clamp (int value , int min , int max);

将给定值限制在由给定最小整数和最大整数值定义的范围之间。如果它在最小值和最大值之间，则返回给定值。

如果给定值小于最小值，则返回最小值。如果给定值大于最大值，则返回最大值。min 和 max 参数包括在内。例如，Clamp(10, 0, 5) 将返回最大参数 5 而不是 4。

11. 最近的二次方- Mathf.ClosestPowerOfTwo

public static int ClosestPowerOfTwo(int value);

返回距离 Value 值最近的 2 的 n 次幂，Mathf.ClosestPowerOfTwo(7)最近值为 8，Mathf.ClosestPowerOfTwo(19)最近值为 16。

12. 余弦- Mathf.Cos

public static float Cos(float f);

f 为输入的角度，以弧度为单位。返回由 f 指定的余弦值，值介于负 1 到 1 之间。

13. 度转弧度- Mathf.Deg2Rad

public static float Deg2Rad;

将角度转换为弧度值，其值等于$\frac{2\pi}{360}$。

14. 弧度转度- Mathf.Rad2Deg

public static float Rad2Deg;

弧度值转换为角度，其值等于$\frac{360}{2\pi}$。

15. 增量角- Mathf.DeltaAngle

public static float DeltaAngle(float current, float target);

计算以度为单位的两个给定角度之间的最短差。

16. 小正数- Mathf.Epsilon

public static float Epsilon;

一个很小的浮点值（只读）。浮点数可以具有的不为零的最小值。

① 规则：

anyValue + Epsilon = anyValue

anyValue - Epsilon = anyValue

0 + Epsilon = Epsilon

0 - Epsilon = -Epsilon

② Example:

```
bool isEqual(float a, float b)
{
    if (a >= b - Mathf.Epsilon && a <= b + Mathf.Epsilon)
    {
        return true;
    }
    else
    {
        return false;
```

```
        }
    }
```

17. 指数- Mathf.Exp

public static float Exp(float power);

反回 e 的 Power 次方的值。

18. 向下最小整数- Mathf.FloorToInt

public static int FloorToInt(float f);

返回小于或等于 f 的最大整数。

Mathf.FloorToInt (10.7f)，返回值为 10。

Mathf.FloorToInt(-10.7f)，返回值为-11。

19. 向下取整- Mathf.Floor

public static float Floor(float f);

返回小于或等于 f 的最大整数。

20. 正无穷- Mathf.Infinity

public static float Infinity;

表示正无穷，也就是无穷大，∞。

21. 反差值- Mathf.InverseLerp

public static float InverseLerp(float a, float b, float value);

计算 value 位于 a 和 b 之间的位置，其值为均匀分布的分布函数，即$\frac{x-a}{b-a}$。

22. 是否为 2 的幂- Mathf.IsPowerOfTwo

public static bool IsPowerOfTwo(int value);

如果该值是 2 的幂，返回 true。

23. 插值角度- Mathf.LerpAngle

public static float LerpAngle(float a, float b, float t);

和 lerp 类似。

24. 插值- Mathf.Lerp

public static float Lerp(float a, float b, float t);

基于浮点数 t，返回 a 到 b 之间的差值，t 限制在 0-1 之间。

当 t 等于 0 时，返回 a 的值；当 t 等于 1 时，返回 b 的值；当 t 等于 0.5 时，返回 a 和 b 的中点。

25. 基数为 10 的对数- Mathf.Log10

public static float Log10(float f);

返回指定数字的以 10 为底的对数。

26. 对数- Mathf.Log

① public static float Log(float f, float p);

返回指定基数中指定数字的对数。

② public static float Log(float f);

返回指定数字的自然对数（以 e 为底）。

27. 最大值- Mathf.Max

public static float Max(float a, float b);

public static float Max(params float[] values);

返回两个或多个值中的最大值。

28. 最小值- Mathf.Min

public static float Min(float a, float b);

public static float Min(params float[] values);

返回两个或多个值中的最小值。

29. 移动角- Mathf.MoveTowardsAngle

public static float MoveTowardsAngle(float current, float target, float maxDelta);

和 MoveTowards 类似。

变量 current 和 target，假定以度为单位。maxDelta 出于优化原因，不支持负值。要 current 远离目标角度，请改为在该角度上增加 180。

30. 移向- Mathf.MoveTowards

public static float MoveTowards(float current, float target, float maxDelta);

将值 current 移向 target。

这与 Mathf.Lerp 基本相同，但该函数将确保移动速度永远不会超过 maxDelta。maxDelta 为负值，将会让 current 远离 target。

31. 负无穷- Mathf.NegativeInfinity

public static float NegativeInfinity;

负无穷，-∞。

32. 下一个 2 的幂- Mathf.NextPowerOfTwo

public static int NextPowerOfTwo(int value);

返回大于或等于该值的，2 的幂。

Mathf.NextPowerOfTwo(7)，返回 8。

Mathf.NextPowerOfTwo(139)，返回 256。

Mathf.NextPowerOfTwo(256)，返回 256。

33. 乒乓- Mathf.PingPong

public static float PingPong(float t, float length);

PingPong 返回一个值，该值将在值 0 和 length 之间来回震荡。

PingPong 要求值 t 是自增值，例如 Time.time 和 Time.unscaledTime。

34. 圆周率- Mathf.PI

public static float PI;

3.1415926

35. 次方- Mathf.Pow

public static float Pow(float f, float p);

返回 f 的 p 次方。

36. 重复- Mathf.Repeat

public static float Repeat(float t, float length);

循环值 t，使其永远不会大于 length 且永远不会小于 0。这类似于模运算符，但它适用于浮点数。例如当 t 为 3.0 且 length 为 2.5，结果将是 0.5。当 t 为 5 且 length 为 2.5，结果将为 0.0。但是请注意，该行为并未像模运算符那样为负数定义。

37. 四舍五入到整数- Mathf.RoundToInt

public static int RoundToInt(float f);

将 f 四舍五入到最接近的整数。

如果数字以 .5 结尾，它介于两个整数之间，其中一个是偶数，另一个是奇数，那么返回偶数。

38. 四舍五入- Mathf.Round

public static float Round(float f);

将 f 四舍五入到最接近的整数。

如果数字以 .5 结尾，它介于两个整数之间，其中一个是偶数，另一个是奇数，那么返回偶数。

39. 有向- Mathf.Sign

public static float Sign(float f);

当 f 为正或 0 时，返回 1；为负时返回-1.

40. 正弦- Mathf.Sin

public static float Sin(float f);

返回 f 所对应的正弦值，返回值介于负 1 到正 1 之间。

41. 平方根- Mathf.Sqrt

public static float Sqrt(float f);

返回 f 的平方根。

42. 正切- Mathf.Tan

public static float Tan(float f);

返回 f 的正切值，f 以弧度为单位。

43. 平滑阻尼角度- Mathf.SmoothDampAngle

public static float SmoothDampAngle(float current, float target, ref float currentVelocity, float smoothTime, float maxSpeed = Mathf.Infinity, float deltaTime = Time.deltaTime);

随着时间的推移逐渐改变当前角度到期望的角度，这个函数可以用来平滑任何一种值、颜色、位置、变量。

44. 平滑阻尼- Mathf.SmoothDamp

public static float SmoothDamp(float current, float target, ref float currentVelocity, float smoothTime, float maxSpeed = Mathf.Infinity, float deltaTime = Time.deltaTime);

随着时间的推移，逐渐改变一个值到目标值。

45. 平滑插值- Mathf.SmoothStep

public static float SmoothStep(float from, float to, float t);

此函数在 Lerp 之间进行插值，min 和 max 以与 Lerp 类似的方式进行插值。但是，插值会从开始逐渐加速，并逐渐减慢到结束。这对于创建淡入淡出和其他过渡非常有用。

六. UGUI 事件接口

命名空间：Using UnityEnging.EventSystems;

1. IPointEnterHandler

当鼠标光标移入该对象时触发。

2. IPointExitHandler

当鼠标光标移出该对象时触发。

3. IPointDownHandler

鼠标点击 A 对象，按下鼠标时 A 对象响应此事件。

4. IPointUpHandler

鼠标点击 A 对象，抬起鼠标时 A 对象响应此事件。(无论鼠标在何处抬起，都会在 A 对象中响应此事件)响应此事件的前提，是 A 对象必须响应过 IPointDownHandler。

5. IPointClickHandler

鼠标点击 A 对象，抬起鼠标时 A 对象响应此事件（鼠标按下和抬起时鼠标要处于同一对象上）。

6. IDragHandler

当鼠标在 A 对象按下并拖拽时，A 对象每帧响应一次此事件。（如果不实现此接口，则后面的四个接口方法都不会触发）

7. IInitializePotentialDragHandler

当鼠标在 A 对象按下还没开始拖拽时，A 对象响应此事件。此接口事件与 IPointDownHandler 接口类似，二者的执行顺序：先执行 IPointDownHandler，然后 执行此接口事件（需先实现 IDragHandler 接口）

8. IBeginDragHandler

当鼠标在 A 对象按下并开始拖拽时，A 对象响应此事件，此事件在 IInitializePotentialDragHandler 之后响应，在 OnDrag 之前响应。

9. IEndDragHanlder

当鼠标抬起时，A 对象响应此事件。

10. IDropHandler

A、 B 对象必须均实现 IDropHandler 接口，且 A 至少实现 IDragHandler 接口。当鼠标从 A 对象上开始拖拽，在 B 对象上抬起时，B 对象响应此事件。此时 name 获取的是 B 对象的 name 属性，eventData.PointerDrag 表示发起拖拽的对象

七. HideFlags

HideFlags 为枚举类，用于控制 Object 对象的销毁方式及其在检视面板中的可视性。

PS：

```
public enum HideFlags
  {
    //默认
    None = 0,
    //不在层级界面显示
    HideInHierarchy = 1,
    //inspector 界面不可见
    HideInInspector = 2,
    //编辑的时候不会被保存
    DontSaveInEditor = 4,
    //inspector 不可以编辑
    NotEditable = 8,
    //构建时不会被保存
    DontSaveInBuild = 16,
    //不能通过 Resources.UnloadUnusedAssets 被卸载
    DontUnloadUnusedAsset = 32,
    //几个拼接
```

```
        DontSave = DontUnloadUnusedAsset | DontSaveInBuild | DontSaveInEditor,
        //几个拼接
        HideAndDontSave = DontSave | NotEditable | HideInHierarchy,
    }
```

PS：

```
public class HideFlagTest : MonoBehaviour
{
    private void Start()
    {
        gameObject.hideFlags = HideFlags.HideInInspector;
        transform.hideFlags = HideFlags.NotEditable;
        GetComponent<BoxCollider>().hideFlags = HideFlags.DontSaveInEditor;
    }
}
```

1. None-默认

None 为 HideFlags 的默认值，即不改变 Object 对象的可见性。

2. HideInHierarchy-不在层级界面显示

此属性的功能是设置 Object 对象在 Hierarchy 面板中是否被隐藏。若隐藏父物体，子物体也会被隐藏掉。隐藏后，物体能看见，但是在 Scene 中无法点击。

3. HideInInspector- inspector 界面不可见

此属性的功能是设置 Object 对象在 Inspector 面板中是否被隐藏。：

① 如果一个对象使用了 HideFlags.HideInInspector，则其所有组件将在 Inspector 中被隐藏，但子类组件依然显示。

② 如果只隐藏对象的某个组件，其他组件不受影响。

4. DontSaveInEditor-编辑的时候不会被保存

这个枚举有个官方说是在编辑模式的时候不会被保存，但是测试会没有效果依旧会记录；另外这里有个很大的坑，使用这个之后物体在场景销毁时候不会跟着销毁，需要使用 DestroyImmediate 手动销毁。

5. NotEditable - 在 inspector 不可以编辑

对象在 Inspector 面板中的可编辑性。

此属性用来设置程序运行时 Object 对象是否可在 Inspector 面板中被编辑。

① 对象使用 HideFlags.NotEditable 可以使得 GameObject 对象的所有组件在 Inspector 面板中都处于不可编辑状态。但 GameObject 对象被 HideFlags.NotEditable 标识并不影响其子类对象的可编辑性。

② 对于 GameObject 对象的某个组件如 Transform 单独使用 HideFlags.NotEditable,只会使得当前组件不可编辑，但 GameObject 的其他组件仍可在 Inspector 面板中编辑。

6. DontSaveInBuild-构建时不会被保存

构建播放器时不保存该对象。使用该标签的对象，我们必须使用 DestroyImmediate 从内存中手动清除该对象以避免内存泄漏。

7. DontUnloadUnusedAsset-不能通过 Resources.UnloadUnusedAssets 被卸载

Resources.UnloadUnusedAssets 不会卸载该对象。我们必须使用 DestroyImmediate 从内

存中手动清除该对象以避免内存泄漏。

8. DontSave-保留对象到新场景

这是 HideFlags.DontSaveInBuild | HideFlags.DontSaveInEditor | HideFlags.DontUnloadUnusedAsset 的快捷方式。

此属性的功能是用来设置是否将 Object 对象保留到新的场景中，如果使用 HideFlags.DontSave，则 Object 对象将在新场景中被保留下来。

① 如果 GameObject 对象被 HideFlags.DontSave 标识，则在新 Scene 中 GameObject 的所有组件将被保留下来，但其子类 GameObject 对象不会被保留到新 Scene 中。

② 不可以对 GameObject 对象的某个组件如 Transform 进行 HideFlags.DontSave 标识，否则无效。

③ 即使程序已经退出，被 HideFlags.DontSave 标识的对象也会一直存在于程序中，造成内存泄漏，对 HideFlags.DontSave 标识的对象，在不需要是或程序退出时需要使用 DestroyImmediate 手动销毁。

9. HideAndDontSave-保留对象到新场景但隐藏

此属性的功能是用来设置是否将 Object 对象保留到新 Scene 中，如果使用 HideFlags.HideAndDontSave，则 Object 对象将在新 Scene 中被保留下来，但不会显示在 Hierarchy 面板中。

八. 扩展方法

1. 为什么要扩展方法

当我们需要重复使用 untiy 没有的函数方法时，反复的编写代码比较麻烦，这时可以使用扩展方法来一次编写，多次使用。

2. 创建一个存储扩展方法的静态类

PS：

```
using UnityEngine;
using System.Collections;

//创建一个包含所有扩展方法的类
//是很常见的做法。此类必须是静态类。
public static class ExtensionMethods
{
    //扩展方法即使像普通方法一样使用，
    //也必须声明为静态。请注意，第一个参数具有“this”关键字，后跟一个 Transform 变量。此变量表示扩展方法会成为哪个类的一部分。
    public static void ResetTransformation(this Transform trans)
    {
        trans.position = Vector3.zero;
        trans.localRotation = Quaternion.identity;
        trans.localScale = new Vector3(1, 1, 1);
    }
}
```

3. 扩展方法使用

如果这个扩展方法包含在命名空间中，则使用时我们也必须带上对应的命名空间。如果没有包含在独立的命名空间中，则可以直接使用。

PS：

```
using UnityEngine;
using System.Collections;

public class SomeClass : MonoBehaviour
{
    void Start () {
        //请注意，即使方法声明中有一个参数，也不会将任何参数传递给此扩展方法。调用此方法的 Transform 对象会自动作为第一个参数传入。
        transform.ResetTransformation();
    }
}
```

4. 注意

正常情况下，我们使用扩展方法时只会使用该静态类下的公共方法。如果我们需要使用到私有方法，则只能通过反射实现。但是，我们不建议在这里去使用反射，因为我们的反射即没有缓存也没有额外的优化，因此在扩展方法中使用反射其性能是非常不好的。