Transform（变换）

属性

```
position:     在世界空间坐标transform的位置。
localPosition:    相对于父级的变换的位置。如果该变换没有父级，那么等同于Transform.position。

eulerAngles:      世界坐标系中的旋转（欧拉角 Vector3）。
localEulerAngles: 相对于父级的变换旋转角度（欧拉角 Vector3）。
rotation:     世界坐标系中的旋转（四元数）。
localRotation:    相对于父级的变换旋转（四元数）。

localScale:   相对于父级的缩放比例。
lossyScale:   全局缩放比例（只读）。

right:    世界坐标系中的自身右方向。//可以赋值
up:   世界坐标系中的自身上方向。//可以赋值
forward:      世界坐标系中的自身前方向。//可以赋值

parent:   父物体的Transform组件。
root:     对象层级关系中的根对象的Transform组件。
childCount:   子物体数量。
```

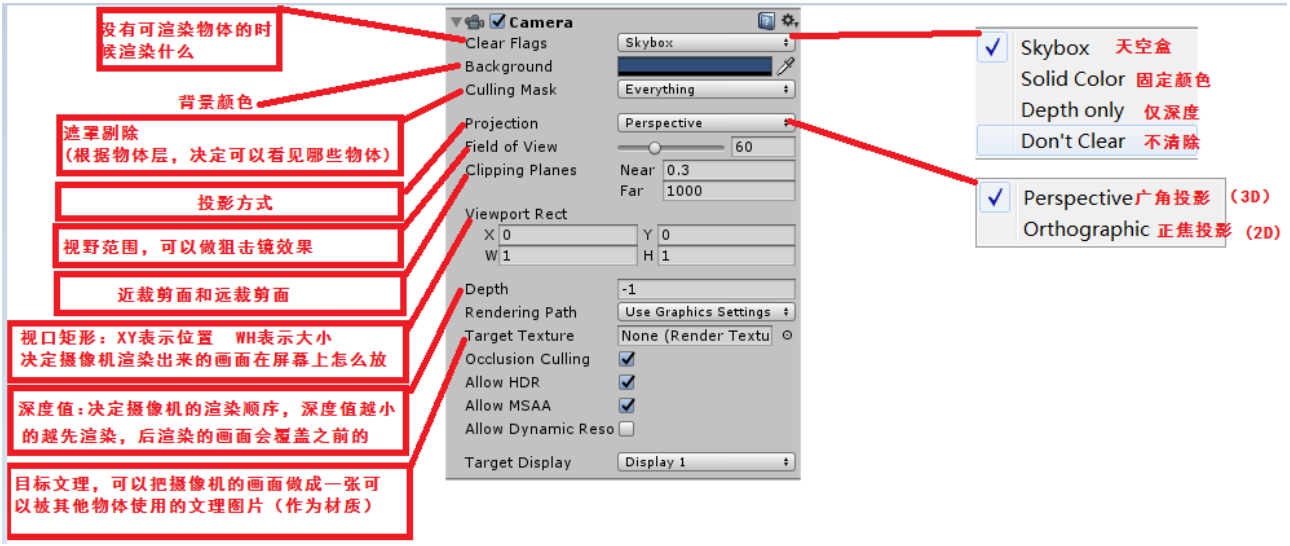
方法

```
Translate();    移动
Rotate();   旋转
RotateAround(); 围绕旋转
SetParent();    设置父物体
GetChild(); 根据索引获取子物体
Find(); 根据名字查找子物体,只能查找一层,可以通过路径找子子物体
GetComponentInChildren();   在子物体中获取组件 包括子子物体 包括自己,不限层数
GetComponentsInChildren();  在子物体中获取组件返回数组  ,同上
TransformDirection();   把相对于世界原点的向量转换成自身方向的向量
TransformPoint();把相对于世界原点的点转换成自身坐标系中的点
SetSiblingIndex();  设置在父物体中索引位置
SetAsFirstSibling();    设置为首个子物体
SetAsLastSibling(); 设置为最后一个
LookAt()看着
```

Time(时间)

```
    Time.time;  游戏开始到现在的总时间
    Time.deltaTime; 每一帧消耗的时间
    Time.fixedDeltaTime;    固定时间(每次物理检测的间隔时间)
    Time.fixedTime; 物理更新总时间
    Time.timeScale; 时间缩放会影响 deltaTime
    Time.unscaledDeltaTime; 未被缩放的   deltaTime
    Time.timeSinceLevelLoad;    当前场景从加载到现在的时间
    Time.frameCount;    从游戏开始到现在的总帧数
```

Camera(摄像机)



渲染路径

Unity支持多种渲染技术，或者称作“路径”。启动项目时需要做出的一个重要的决定就是使用哪条路径。 Unity的默认值为“Forward Rendering”。

Forward Rendering

在“向前渲染（Forward Rendering）”中，每个对象对于影响它的每个光线都以“通过（pass）”形式呈现。因此，每个对象可能会被渲染多次，具体取决于范围内有多少个灯。

这种方法的优点是非常快，这意味着它的硬件要求低于延迟渲染（Deffered Rendering）。另外，向前渲染（Forward Rendering）为我们提供了可定制的“阴影模型”，可以快速处理透明度。它还允许使用例如”multi-sample anti-aliasing”（MSAA）之类的硬件技术，这些技术在延迟渲染（Deferred Rendering）中不可以使用。

然而，一个显着缺点是我们必须支付每个光源的渲染成本。也就是说，影响对象的光线越多，渲染性能越慢。

但是，如果可以在游戏中管理灯光的数量，则向前渲染（Forward Rendering）是一个非常快速的解决方案。

Deferred Rendering

这种方法的主要优点在于，照明的渲染成本与光照射的像素数成正比，而不是灯本身的数量。因此，不再受屏幕上要渲染的灯光数量的限制，对于某些游戏来说，这是一个关键的优势。
延迟渲染（Deferred Rendering）提供了可预测的性能，但通常需要更强大的硬件。某些移动硬件也不支持此功能。

作业:

>1每间隔1秒生成一个方块,间隔时间可以改
>2做一个狙击镜效果
>3WS控制移动,AD控制左右转,空格键向前方发射子弹