【Unity基础教程】重点知识汇总

（八）

Unity中Transform组件常用属性及方法

Transform组件

概念：在Unity中，Transform组件是每个游戏对象（如场景中的角色、摄像机、灯光等）的**核心部分**，用于控制游戏对象的位置（**Position**）、旋转（**Rotation**）和缩放（**Scale**）。Transform是一个非常重要的类，它提供了许多**属性**和**方法**，可以轻松操作对象的空间变换。

position（属性）

含义及用法：表示游戏对象在**世界空间**中的位置**（Vector3）**，获取或设置物体的位置（坐标）。

```
1  void Start()
2  {
3      // 将对象移动到世界坐标 (0, 1, 0)
4      transform.position = new Vector3(0, 1, 0);
5  }
```

localPosition（属性）

含义及用法：表示游戏对象在其**父对象**的**本地坐标**空间中的位置，即相对于父物体的位置（**Vector3**）。如果物体没有父物体，则等同于position。同理，如果对象有父对象，localPosition可以理解为是相对于父对象位置的偏移量。

```
1 // 设置对象在父对象本地坐标中的位置
2 transform.localPosition = new Vector3(1, 0, 0);
```

rotation（属性）

含义及用法：表示对象在**世界空间**的旋转，使用的是**四元数（Quaternion）**，避免了欧拉角旋转中的**万向节锁**问题。如果你更习惯使用欧拉角，可以用**Quaternion.Euler**来创建四元数。

```
1  // 设置对象绕Y轴旋转90度
2  transform.rotation = Quaternion.Euler(0, 90, 0);
```

localRotation（属性）

含义及用法：表示对象在**本地坐标**空间中的旋转，也使用**四元数（Quaternion）**。localRotation是对象相对于父对象的旋转。（同样，也可以用**Quaternion.Euler**来创建四元数。

```
// 设置本地旋转为45度
transform.localRotation = Quaternion.Euler(0, 45, 0);
```

*四元数与欧拉角（概念）

四元数：在Unity和计算机图形学中，四元数（Quaternion）是一种高效且稳定的表示3D旋转的数学工具，主要用于**避免欧拉角旋转中的万向节锁（Gimbal Lock）问题，并提供平滑插值功能。**是一种扩展复数的数学结构，用于表示旋转。它由一个**标量**部分和一个**向量**部分组成。写作：**Q = w + xi + yj + zk**。

- 其中 w 是标量部分（实部），(x,y,z) 是向量部分（虚部），i,j,k是虚数单位（满足$i^2=j^2=k^2=ijk=-1$）。
- 在Unity中，四元数通常表示为**Quaternion(w, x, y, z)**，或者更常见地用Quaternion.eulerAngles表示欧拉角。

在3D图形中，四元数的核心作用是表示旋转，它可以看作是围绕一个**单位向量** (x, y, z) 旋转一个角度θ的变换。

*四元数与欧拉角（概念）

四元数（续）： 四元数支持球面线性插值（Slerp）和其他插值方法，能够实现平滑旋转过渡，而欧拉角容易产生不连续的跳跃。同时使用四元数进行旋转计算比旋转矩阵更高效，占用的内存也更少。

方法	作用
Quaternion.Euler()	将欧拉角转换为四元数。
Quaternion.AngleAxis()	创建一个围绕指定轴旋转的四元数。
Quaternion.identity	返回一个无旋转的四元数（w=1，x=0，y=0，z=0）。
Quaternion.Inverse()	返回四元数的逆，用于反向旋转。
Quaternion.LookRotation()	创建一个面向指定方向的四元数，通常用于让对象朝向目标。
Quaternion.Lerp()	线性插值，用于平滑旋转，但速度可能不均匀。
Quaternion.Slerp()	球面线性插值，用于平滑旋转，速度均匀。
Quaternion.Angle()	计算两个四元数之间的角度。

Unity中的Quaternion类封装了四元数的创建、计算和变换操作。

*四元数与欧拉角（概念）

欧拉角：欧拉角（Euler Angles）是用于描述三维空间中旋转的一种方式，它由三个角度组成，表示对象绕固定坐标轴（通常是X、Y、Z轴）旋转的角度。欧拉角在Unity和3D编程中非常常见，因为它直观且易于理解。欧拉角的核心思想是通过三个旋转角度来描述一个物体的旋转，**绕X轴旋转（称为Pitch或俯仰角），绕Y轴旋转（称为Yaw或偏航角），绕Z轴旋转（称为Roll或滚转角）**。这些旋转是**逐轴旋转**，即每次先绕一个轴旋转，接着绕第二个轴旋转，再绕第三个轴旋转。欧拉角的顺序（即哪个轴先旋转）非常重要，因为不同的旋转顺序可能会得到不同的结果。（在Unity中通常用**Vector3三维向量**来表示欧拉角）

```
// 将物体旋转到绕Y轴90度的位置
transform.eulerAngles = new Vector3(0, 90, 0);
```

*四元数与欧拉角（概念）

欧拉角（续）：万向节锁是欧拉角表示法的一个缺点。当物体的两个旋转轴（例如，俯仰角和偏航角）重合时，旋转自由度会丧失，导致系统无法进行某些旋转。具体来说，万向节锁发生在某些角度（例如，当俯仰角接近90°或-90°时），此时会**失去一个自由度（指旋转轴重合、失去独立控制或无法达到预期旋转效果），导致旋转行为变得不连续或难以控制**。

- 例如，如果物体围绕 X 轴旋转了90°，则绕 Y 轴和 Z 轴的旋转变得不再独立，系统会失去一个自由度，无法继续执行预期的旋转。

欧拉角在旋转插值（如从一个方向平滑过渡到另一个方向）时，可能会出现不连续的跳跃。

- 例如，Vector3(0, 360, 0) 和 Vector3(0, 0, 0) 在数学上是相同的方向，但插值时会产生大角度旋转。
- 四元数可以通过球面插值（Slerp）解决这一问题。

*四元数与欧拉角（区别）

欧拉角与四元数的对比

特性	欧拉角（Euler Angles）	四元数（Quaternion）
表示	以三个角度（X、Y、Z）表示旋转。	以四元数 `w + xi + yj + zk` 表示旋转。
直观性	可被人类直观理解，易于调试和编辑。	数学表示不直观，难以手动调试。
存储空间	占用更少的内存（3个值）。	占用更多的内存（4个值）。
万向节锁问题	会产生万向节锁，导致旋转自由度丢失。	不存在万向节锁问题。
插值	插值可能不连续，容易发生跳跃。	支持平滑插值，适合动画中的旋转过渡。
旋转顺序依赖	旋转结果依赖于轴的旋转顺序，容易导致混淆。	不依赖旋转顺序，旋转结果唯一。
使用场景	适合简单的旋转操作，或者需要手动编辑旋转值时使用。	适合复杂的连续旋转，如动画和物理计算。

在Unity中，四元数是底层旋转的核心，但欧拉角作为一个直观的接口让开发者更容易操作旋转。对于简单场景，直接使用欧拉角即可，而对于复杂的旋转和插值，建议使用四元数（相互转化）。

```
// 从欧拉角转换为四元数
Vector3 eulerAngles = new Vector3(30f, 45f, 60f);
Quaternion quaternion = Quaternion.Euler(eulerAngles);
// 从四元数转换为欧拉角
Quaternion quaternion = Quaternion.Euler(30f, 45f, 60f);
Vector3 eulerAngles = quaternion.eulerAngles;
```

localScale（属性）

含义及用法：表示对象的缩放比例（Vector3），是相对于**父对象**的缩放。如果父对象的缩放是非默认值（1,1,1），子对象的缩放会受到影响。

```
// 将对象放大2倍
transform.localScale = new Vector3(2, 2, 2);
```

up、right、forward（属性）

含义及用法：up：获取物体的“上”方向（通常是**Y**轴方向），right：获取物体的“右”方向（通常是**X**轴方向），forward：获取物体的“前”方向（通常是**Z**轴方向）。均为**单位向量**，会随着对象的旋转而变化，是一个**动态计算**的值（反方向为负值）。

```
1 // 获取物体的上方向
2 Vector3 up = transform.up;
3 // 获取物体的右方向
4 Vector3 right = transform.right;
5 // 获取物体的前方向
6 Vector3 forward = transform.forward;
```

parent（属性）

含义及用法：表示对象的**父对象**（Transform），将父对象设置为null可以将对象移到场景的根层级，即脱离父子物体关系。

```
// 获取父对象
Transform parentTransform = transform.parent;
// 设置新的父对象
transform.parent = null;
```

childCount（属性）

含义及用法：获取物体的子物体数量。

```
1  // 获取子物体数量
2  int count = transform.childCount;
```

Translate（方法）

含义及用法：移动对象，**Translate(Vector3 translation, Space relativeTo = Space.Self)**，默认情况下，移动是基于对象的**本地坐标**。（相对于当前物体的坐标系来平移物体。可以指定平移的方向和距离，支持世界坐标系和本地坐标系）其中，Vector3表示目标平移的**方向**和**距离**，Space指定平移是否使用世界坐标系或本地坐标系。

```
1  // 向前平移 5 单位
2  transform.Translate(Vector3.forward * 5);
3  // 在世界空间中向上平移 3 单位
4  transform.Translate(Vector3.up * 3, Space.World);
```

Rotate（方法）

含义及用法：使用**欧拉角**旋转对象，**Rotate(Vector3 eulerAngles, Space relativeTo = Space.Self)**，旋转的角度是基于物体当前的方向。Vector3表示旋转的角度（以**度**为单位），Space指定旋转是基于世界坐标系还是本地坐标系。

```
1  // 默认在本地坐标空间中绕Y轴旋转90度
2  transform.Rotate(new Vector3(0, 90, 0));
3  // 在世界坐标中旋转
4  transform.Rotate(new Vector3(0, 90, 0), Space.World);
```

RotateAround（方法）

含义及用法：使对象围绕指定的点和轴进行旋转。**RotateAround(Vector3 point, Vector3 axis, float angle)**，point为旋转的中心点，表示对象要围绕的点的世界空间坐标（可以表示自身）。axis表示旋转的轴（旋转的方向，为**单位向量**），即物体围绕哪个轴旋转（比如Vector3.up或(0, 1, 0)表示围绕Y轴旋转）。angle表示旋转的角度，单位为度。正值表示顺时针旋转，负值表示逆时针旋转。

```
1 private void Update()
2 {
3     // 围绕目标的世界位置，绕Y轴旋转，每秒旋转 speed 度
4     // speed * Time.deltaTime 确保旋转是基于时间的，旋转速度不会因帧率不同而变化。
5     transform.RotateAround(target.position, Vector3.up, speed * Time.deltaTime);
6     // 物体自身的中心点作为旋转中心，绕Y轴旋转
7     transform.RotateAround(transform.position, Vector3.up, 1f);
8 }
```

LookAt（方法）

含义及用法：让当前对象（**向前的方向，一般为Z轴正方向**）面向目标对象，目标对象可以是一个位置（Vector3）或一个目标物体（Transform）。**LookAt(Transform target)**，Target表示要朝向的目标位置或物体。可以用重载方法指定“向上的方向”（默认为 (0,1,0)）。

```
1  // 让当前对象面向目标对象
2  transform.LookAt(target, new Vector3(0, 1, 0));
```

SetParent（方法）

含义及用法：将物体设置为另一个物体的子物体。**SetParent(Transform parent, bool worldPositionStays = true)**，parent表示新的父物体，worldPositionStays表示是否保持世界坐标不变，如果为true，则物体会保持其世界坐标；如果为false，则会调整物体的本地坐标。

```
// 设置父物体
transform.SetParent(parentTransform);
// 设置父物体，并保持局部坐标不变
transform.SetParent(newParentTransform, false);
```

DetachChildren（方法）

含义及用法：移除当前对象下的所有子对象，**DetachChildren()**，从父物体中分离所有子物体（即取消其父物体的关联）。

```
1  // 将所有子对象从当前对象中移除
2  transform.DetachChildren();
```

Find（方法）

含义及用法：通过名字查找当前物体下的某个子物体。**Find(string name)**，name表示需要查找的子物体的名称，如果找到了该子物体，则返回其Transform，否则返回null。

```
1  // 查找名为ChildName的子对象
2  Transform child = transform.Find("ChildName");
```

GetChild（方法）

含义及用法：按**索引**获取子对象。**GetChild(int index)**。

```
1  // 获取第一个子对象
2  Transform child = transform.GetChild(0);
```

【Unity基础教程】重点知识汇总

（八）

Unity中Transform组件常用属性及方法