计算机系科协暑培 5.0

Unity 场景与脚本基础

计算机系学生科协 谢语桐

2024 年 8 月 6 日

本讲内容

① 场景

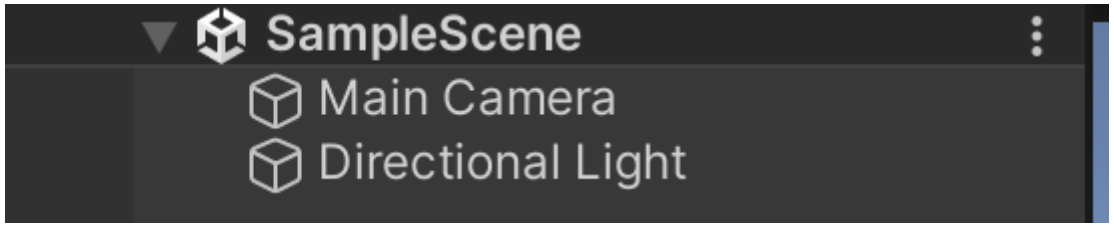
- 基本概念
- 坐标系统
- 相机

② 脚本基础

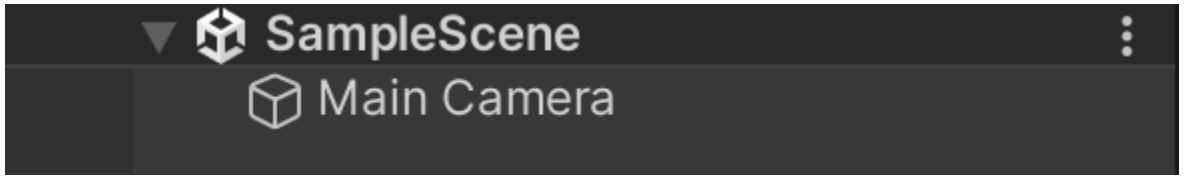
- 执行顺序
- 作为组件的简单应用
- 常用 API 示例

场景的基本概念

Unity 中的场景是一个虚拟的 2D 或 3D 环境，用于放置、组织和展示游戏对象、视觉效果等元素。



创建 3D 项目时自带的默认场景



创建 2D 项目时自带的默认场景

场景的基本概念

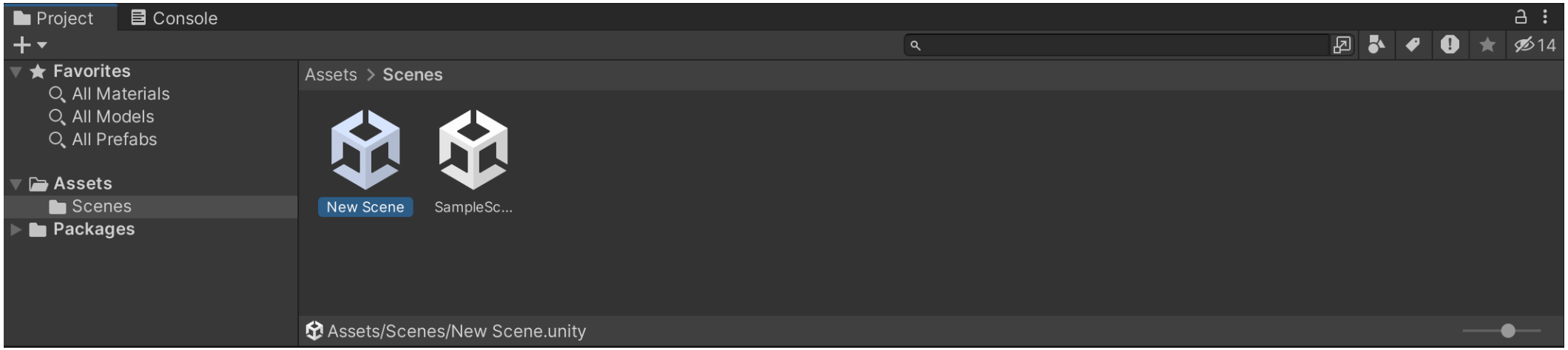
我们可以在场景中创建各种游戏对象，并给它们挂载一些组件。

Unity 中的每个场景都对应了磁盘上的一个 .unity 格式文件，这个文件存储了场景中游戏对象的信息，包括它们挂载的组件。

需要注意的是，场景文件通过一串 hash 码识别各个游戏对象和组件，因此在团队开发时，有可能因为不同开发者电脑上的 hash 码不同而导致合并冲突、场景错乱等问题。我们将在后续的“项目结构与项目管理”这一课程中详细说明这个问题。

场景的基本概念

可以在下方的 Project 面板中新建场景。

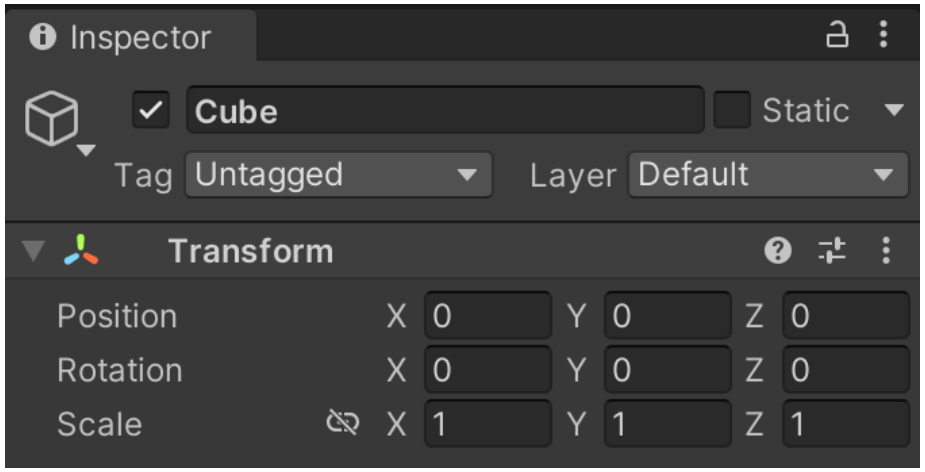


新建的场景

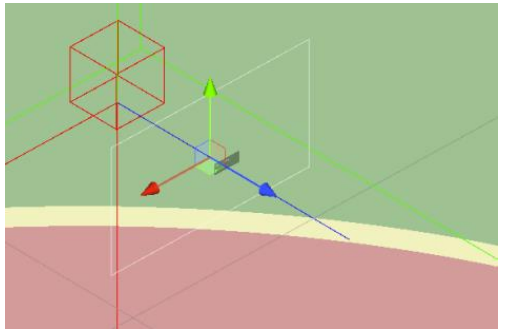
在游戏运行时可以实现多个场景之间的切换。具体切换方法在本节课的后续内容中详细说明。

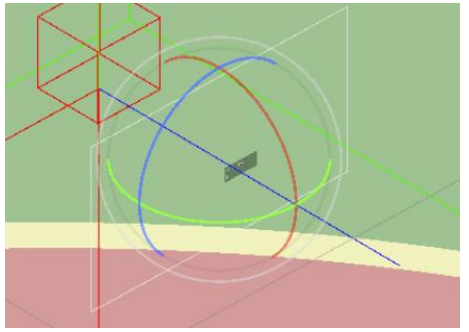
坐标系统 – 概述

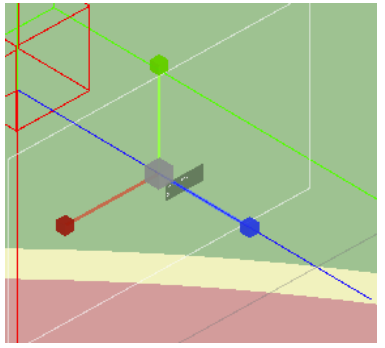
Unity 以物体的中心坐标、旋转和缩放唯一确定其各个顶点的位置。我们可以在物体的 Transform 组件中看到这些信息（每个游戏对象都会默认地挂载 Transform 组件；在 UI 对象中是 Rect Transform 组件）。



Transform 组件







利用 Unity 编辑器修改这三个属性

我们可以在 Unity 编辑器中方便地修改这三个属性。

坐标系统 - 概述

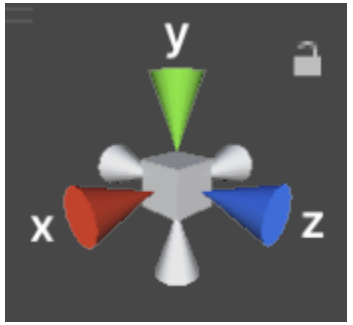
概念回顾：每个被挂载的组件其实都是一个类的对象。因此，图中的 Position、Rotation 和 Scale 都是 Transform 类的成员。这样一来我们不仅可以在编辑器中手动修改这些数值，还可以通过编写代码来实现修改。

三种属性在 Transform 中对应的变量名：

- 中心坐标：三维矢量(.position)
- 旋转：四元数(.rotation)
- 缩放：三维矢量(.lossyScale)

坐标系统 – 绝对坐标

对于每个场景，Unity 定义了一个世界坐标系。

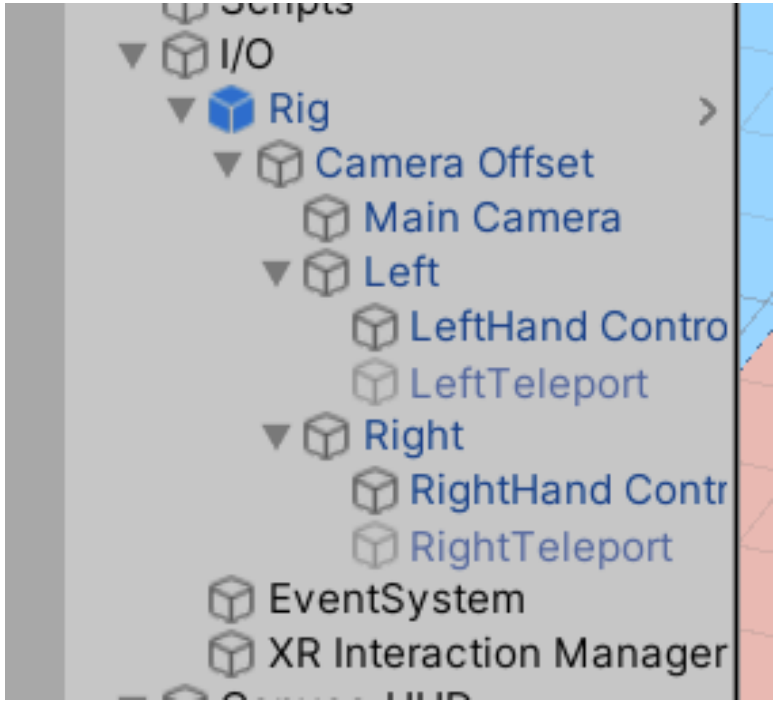


Unity 中的坐标系图示

需要注意的是，纵向坐标被表示为 y 轴，而与之垂直的平面是 x0z。（以 x, y, z 的顺序来看，该坐标系似乎是一个左手系）

坐标系统 – 相对坐标

GameObject 嵌套：Unity 可以在一个 GameObject 内嵌套多个其他 GameObject。



GameObject 嵌套

坐标系统 - 相对坐标

在 GameObject 嵌套中，子物件的 Transform 组件在编辑器中显示出来的数值是相对其父物件的坐标，这称为相对坐标。如果一个物件没有父物件（即其父物件为场景），那么显示出来的是绝对坐标。

三种相对坐标在 Transform 中对应的变量名：

- 中心坐标：三维矢量(.localPosition)
- 旋转：四元数(.localRotation)
- 缩放：三维矢量(.localScale)

坐标系统 – 平移和旋转

利用代码，我们可以实现物体的平移和旋转。

设平移量为三维向量 v，平移的代码实现为：

```
transform.Translate(v);
```

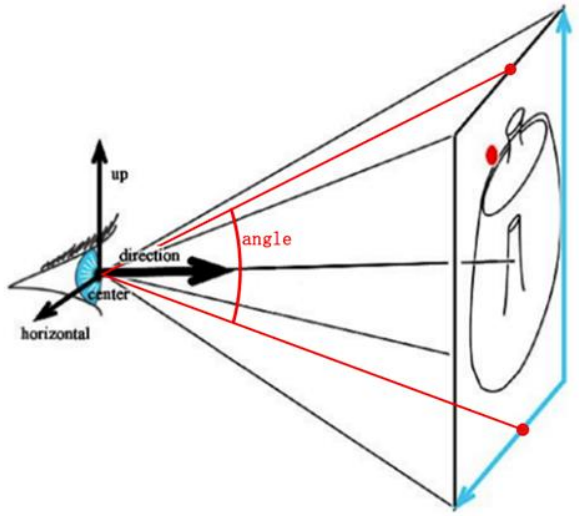
对于旋转，由于四元数的概念较为复杂，我们可以直接利用 Unity 提供的接口进行操作：

```
transform.RotateAround(point,axis,angle);
```

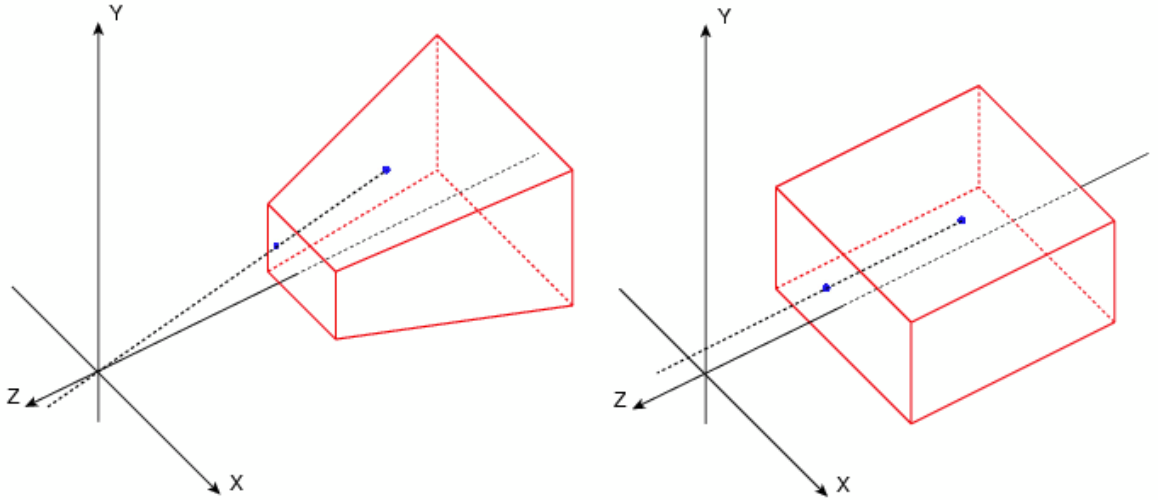
其中，point 为旋转轴穿过的点；axis 为旋转轴；angle 为旋转角。

相机

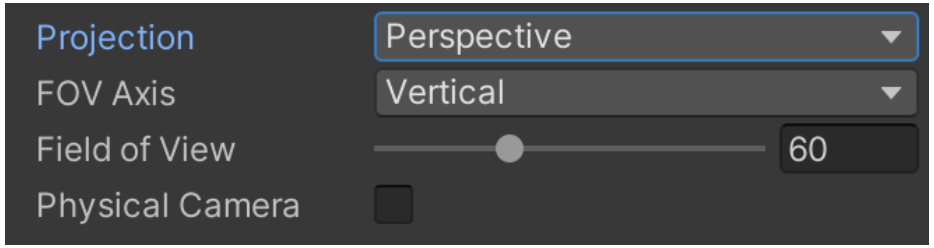
Unity 中相机有两种类型：正交相机和透视相机。

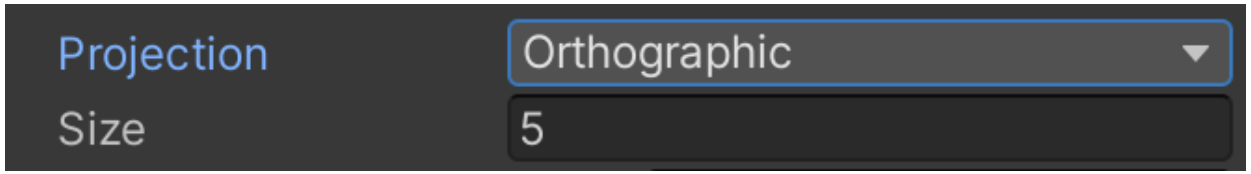


透视相机 – 近大远小



正交相机 – 没有透视效果





相机

多个相机情况下，常用的属性：Viewport Rect 和 Depth。

Viewport Rect

X 为绘制摄像机视图的起始水平位置；Y 为绘制摄像机视图的起始垂直位置；W 为屏幕上摄像机输出的宽度；H 为屏幕上摄像机输出的高度。这四个值均在 0 到 1 之间，表示百分比。

Depth

相机的深度(Depth)控制场景中不同相机的渲染顺序。深度值越小，相机的渲染顺序越靠前，渲染出的画面作为背景；而深度值越大，相机的渲染顺序越靠后，渲染出的画面作为前景。

执行顺序

Unity 中有许多类似于“事件检测”的 API。在一次完整的游戏中，这些 API 的具体执行顺序如下：

Awake -> OnEnable -> Start -> OnXXX -> Update -> OnDisable -> OnDestroy

- Awake：GameObject 被初始化而且启用后立即调用，用于一些非常重要的初始化（血量等）。但大多数时候用 Start 足矣。
- OnDestroy：物体被销毁时候的行为。
- OnEnable / OnDisable：挂载在 GameObject 上的脚本组件被启用、停用时的行为。
- Start："Start is called before first frame update."（注意：Unity 会在执行完所有脚本的 Start 之后再进入更新周期）

执行顺序

Unity 中有许多类似于“事件检测”的 API。在一次完整的游戏中，这些 API 的具体执行顺序如下：

Awake -> OnEnable -> Start -> OnXXX -> Update -> OnDisable -> OnDestroy

- Update：以一定的时间跨度（帧率），反复调用该函数，常用于更新游戏元素的物理状态（如位移、金币、布局、鼠标键盘事件）
- OnXXX：监听事件。例如 OnCollisionEnter、OnMouseDown 等。可以理解为 Unity 在反复调用 Update 之间，插入对事件监听的处理。

作为组件的简单应用

一个组件可以通过声明 public 成员来将该成员显示到 unity 编辑器中，从而可以在 unity 编辑器中对其设置初值。

利用这个方法，我们可以方便地设置一些属性的初值。例如一个小球的移动速度（声明时不能直接赋值，在start中专门赋值又累赘）；此外，也可以用这个办法来预设将要复制的游戏对象（本节课后续内容）。

此外，也可以用相同的组件构造不同的预制体，比如一个学生有男生和女生两种，那么就可以添加一个公共的 gender 属性，只不过一个选男，一个选女，分别保存成预制体，就可以有一个男生预制体和一个女生预制体了。

脚本可以作为一种“组件”挂载到游戏对象上

常用 API 示例

Unity 有很多 API，使用起来非常灵活方便，功能也很强大。实际开发过程中，通常可以查阅官方脚本 API 文档：

https://docs.unity.cn/cn/current/ScriptReference/index.html

本节课主要讲几个用得比较多的例子，便于同学们快速上手：

- 获取游戏对象和组件
- 复制游戏对象（预制体）
- 捕获鼠标和键盘动作
- 切换场景
- 延时调用和循环调用
- 计时器

常用 API 示例

- 获取游戏对象和组件

GameObject.Find

public static GameObject **Find** (string **name**);

描述

按 `name` 查找 GameObject，然后返回它。

此函数仅返回活动 GameObject。如果未找到具有 `name` 的 GameObject，则返回 null。如果 `name` 包含“/”字符，则会向路径名称那样遍历此层级视图。

出于性能原因，建议不要每帧都使用此函数，而是在启动时将结果缓存到成员变量中，或者使用 GameObject.FindWithTag。

- 它有很多“变种”，如 FindWithTag（按标签搜索）、Transform.Find（搜索子 GameObject）等。

常用 API 示例

- 获取游戏对象和组件

Component.GetComponent

public Component **GetComponent** (Type **type**);

- 我一般习惯用 GetComponent<T>()，但它还有很多不同的调用方式，详见文档。
- 作用是获取名字叫 T 的组件。可以这样用：

```
Transform test = obj.GetComponent<Transform>();
Debug.Log(test.name);
```

- 这样会输出 GameObject obj 的名称。如果直接调用 GetComponent，那么会尝试在这个脚本被挂载到的游戏对象上获取指定名称的组件。

常用 API 示例

- 复制游戏对象（预制体）

Object.Instantiate

public static Object **Instantiate** (Object **original**);

public static Object **Instantiate** (Object **original**, Transform **parent**);

public static Object **Instantiate** (Object **original**, Transform **parent**, bool **instantiateInWorldSpace**);

public static Object **Instantiate** (Object **original**, Vector3 **position**, Quaternion **rotation**);

public static Object **Instantiate** (Object **original**, Vector3 **position**, Quaternion **rotation**, Transform **parent**);

- 在当前场景中复制一个 GameObject original，并返回被复制出来的那个对象。
- 这个 API 有超多重载，详见文档。

常用 API 示例

- 捕获鼠标和键盘动作

Input.GetKey
在用户按下 name 标识的键时返回 true。

Input.GetKeyDown
在用户开始按下 name 标识的键的帧期间返回 true。

Input.GetKeyUp
在用户释放 name 标识的键的帧期间返回 true。

- 这三个函数作用都是捕获键盘的动作。区别在于第一个是按下期间会一直返回 true，第二个是只有按下的那一帧返回 true，第三个是只有松开的那一帧返回 true。

常用 API 示例

- 捕获鼠标和键盘动作

Input.GetAxis

public static float **GetAxis** (string **axisName**);

- 捕获鼠标、手柄摇杆和键盘方向键的动作。详见文档。
- 对于鼠标，当鼠标向右或向下移动时，该函数返回值为正；向左或向上移动时，该函数返回值为负。鼠标移动越快，这个函数返回值的绝对值就越大。

常用 API 示例

- 切换场景

SceneManager.LoadScene

public static void **LoadScene** (int **sceneBuildIndex**, SceneManagement.LoadSceneMode **mode**= LoadSceneMode.Single);

public static void **LoadScene** (string **sceneName**, SceneManagement.LoadSceneMode **mode**= LoadSceneMode.Single);

SceneManager.LoadSceneAsync

public static AsyncOperation **LoadSceneAsync** (string **sceneName**, SceneManagement.LoadSceneMode **mode**= LoadSceneMode.Single);

public static AsyncOperation **LoadSceneAsync** (int **sceneBuildIndex**, SceneManagement.LoadSceneMode **mode**= LoadSceneMode.Single);

public static AsyncOperation **LoadSceneAsync** (string **sceneName**, SceneManagement.LoadSceneParameters **parameters**);

public static AsyncOperation **LoadSceneAsync** (int **sceneBuildIndex**, SceneManagement.LoadSceneParameters **parameters**);

- 加载指定索引号的场景或指定名称的场景。

常用 API 示例

- 延时调用和循环调用

MonoBehaviour.Invoke

public void **Invoke** (string **methodName**, float **time**);

- 在 time 秒后调用一次名为 methodName 的方法。

MonoBehaviour.InvokeRepeating

public void **InvokeRepeating** (string **methodName**, float **time**, float **repeatRate**);

- 在 time 秒后调用一次名为 methodName 的方法，随后每隔 repeatRate 秒调用一次。
- 可以通过 IsInvoking 方法来判断是否有未处理的 Invoke、通过 CancelInvoke 方法来取消 Invoke 调用。

常用 API 示例

- 计时器

Time.deltaTime

public static float **deltaTime** ;

- 从上一帧到现在过去的时间，单位为秒。
- 利用这个量可以实现计时器。

答疑时间

谢谢