

TURING

图灵原创

Unity API 解析

陈泉宏◎编著

理解Unity核心API

- 快速提升开发能力



人民邮电出版社

POSTS & TELECOM PRESS

TURING 图灵原创

014061584

TP317.6
15

Unity API 解析

陈泉宏◎编著



北航

C1748051

人民邮电出版社
北京

TP317.6
15

014081284

图书在版编目 (C I P) 数据

Unity API解析 / 陈泉宏编著. — 北京 : 人民邮电出版社, 2014.9
(图灵原创)
ISBN 978-7-115-36651-1

I. ①U… II. ①陈… III. ①游戏程序—程序设计
IV. ①TP311.5

中国版本图书馆CIP数据核字(2014)第171975号

内 容 提 要

本书挑选了Unity引擎里一些核心API类,例如Object、GameObject、Rigidbody、Transform、Camera、Quaternion、Vector3等进行了详细的功能注解,注解内容包括API的使用方法、算法分析、边界条件、参数间的制约关系及注意事项等,特别是对很多功能相近或使用方法相似的API进行了较为详细的比较说明。本书适用于对Unity有一定了解的入门开发人员,也可作为Unity开发者的参考手册。

-
- ◆ 编 著 陈泉宏
责任编辑 王军花
执行编辑 张 霞
责任印制 焦志炜
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京鑫正大印刷有限公司印刷
 - ◆ 开本: 800×1000 1/16
印张: 16
字数: 409千字
印数: 1-3 000册
- 2014年9月第1版
2014年9月北京第1次印刷
-

定价: 49.00元

读者服务热线: (010)51095186转600 印装质量热线: (010)81055316

反盗版热线: (010)81055315

广告经营许可证: 京崇工商广字第 0021 号

前 言

从我开始接触 Unity 到现在已经两年有余。记得刚开始学习 Unity 的时候甚是惊喜，一是因为 Unity 具有强大的多平台移植功能，随着移动互联时代的到来，该功能越来越顺应时代潮流；二是因为这个引擎容易上手，3 个月的时间足以让一个新手对这套引擎有全面的了解。然而，随着时间的推移，我越来越感觉自己的能力提升似乎遇到了瓶颈。在实际的项目开发过程中，总有一种放不开手脚的感觉，就像自己手里明明有一把屠龙刀，却无法发挥出这把刀真正的威力。Unity 是个人门快、提高难的游戏引擎，想提升能力，至少需要越过 3 道坎：第一道坎就是对 API 的积累，对 API 的合理利用不仅可以减轻自己的编码负担，而且往往可以提高程序的运行效率；第二道坎是 shader 编程，想要做出一款精品游戏往往需要有高效 shader 的支持；最后一道坎就是综合能力了，一款产品的制作除了功能编程外，往往会涉及很多其他领域，例如产品架构、UI 交互设计、模型制作等，作为主要的编程人员，对其他相关领域的了解程度往往会影响到产品的制作直至最后的产品体验。

目前，Unity 引擎的 API 有好几千个，并且随着 Unity 版本的更新，API 的数量还会不断增长。对 API 的熟悉程度直接影响着程序的开发效率，熟悉 API 也成了新手进阶的必经之路。尽管官方给出了较为丰富的 API 文档，然而这并不能满足实际开发的需要，因为官方给出的 API 解释往往只描述了相应 API 的主要功能，缺少对其边界条件的说明和 API 的算法解释。要知道，在实际开发中，必须要明确 API 参数的使用范围，否则往往会出现一些无法预料的情况。于是我决定自己来研究。对 API 的研究几乎占用了我所有的业余时间，在这期间我遇到了各种各样的难题，解决后做了详细的笔记。后来，当笔记积累了厚厚一本的时候，我意识到，也许应该分享出来，使更多的开发者在这上面少花费些时间。于是便有了这本书，希望能对开发者有所帮助。

阅读说明

在本书中，以下专业用语的意义是相同的：“变量”与“属性”，“函数”与“方法”，“归一化”与“单位化”。

本书对 API 的描述主要分为 3 个部分：“基本语法”、“功能说明”和“实例演示”。其中，“基本语法”中也包括对 API 参数的简单说明；“功能说明”是对 API 功能的详细介绍，例如使用的注意事项、边界条件、算法分析等；“实例演示”主要是结合实际例子来说明 API 的使用，为了节省篇幅，有些示例代码中会涉及多个 API 的说明。

除了对单个 API 进行解释之外,本书还对一些功能相近或容易混淆的 API 进行了相应的解释。另外,书中所涉及的 API 均基于 Windows 操作系统下 Unity 4.3 版本进行了测试。

源码说明

本书所有“实例演示”的源码都可以在图灵社区本书主页(<http://www.ituring.com.cn/book/1474>)免费注册下载。书中的每个类为一个单独工程,每个工程中场景的命名规则是:API 名字_unity。如果想查看某个 API 的示例代码,只要去 API 所在的类对应的工程文件里打开 API 名字对应的场景即可。例如,打开 Matrix4x4 类中 MultiplyVector 的实例演示的过程如下所示。



致 谢

这本书得以出版，首先要感谢父母和小妹对我的理解和支持，其次要感谢那些在 API 研究过程中为我解惑的朋友们，最后要特别感谢图灵的编辑王军花和张霞为本书出版提供的真诚指导。

由于时间和能力有限，本书只对 Unity 中的 14 个类进行了讲解和说明，难免有不足或疏漏之处，希望能够抛砖引玉，与大家共同研讨。如有疑问或建议，请发邮件至：unity3d_api@163.com。最后，祝愿图灵教育为 IT 行业贡献更多有价值的图书！

目 录

第 1 章 Application 类	1
1.1 Application 类静态属性	1
1.1.1 dataPath 属性: 数据文件 路径	1
1.1.2 loadedLevel 属性: 关卡索引	2
1.2 Application 类静态方法	4
1.2.1 CaptureScreenshot 方法: 截屏	4
1.2.2 LoadLevelAdditiveAsync 方法: 异步加载关卡	11
1.2.3 RegisterLogCallback 方法: 注册委托	12
第 2 章 Camera 类	14
2.1 Camera 类实例属性	14
2.1.1 aspect 属性: 设置摄像机视口 比例	14
2.1.2 cameraToWorldMatrix 属性: 变 换矩阵	15
2.1.3 cullingMask 属性: 摄像机按 层渲染	17
2.1.4 eventMask 属性: 按层响应 事件	18
2.1.5 layerCullDistances 属性: 层消隐的距离	20
2.1.6 layerCullSpherical 属性: 基 于球面距离剔除	21
2.1.7 orthographic 属性: 摄像机投 影模式	22
2.1.8 pixelRect 属性: 摄像机渲染 区间	23
2.1.9 projectionMatrix 属性: 自定 义投影矩阵	25
2.1.10 rect 属性: 摄像机视图的位 置和大小	27
2.1.11 renderingPath 属性: 渲染 路径	29
2.1.12 targetTexture 属性: 目标 渲染纹理	30
2.1.13 worldToCameraMatrix 属性: 变换矩阵	32
2.2 Camera 类实例方法	32
2.2.1 RenderToCubemap 方法: 生成 Cubemap 静态贴图	33
2.2.2 RenderWithShader 方法: 使用 其他 shader 渲染	34
2.2.3 ScreenPointToRay 方法: 近视 口到屏幕的射线	35
2.2.4 ScreenToViewportPoint 方法: 坐标系转换	36
2.2.5 ScreenToWorldPoint 方法: 坐标系转换	37
2.2.6 SetTargetBuffers 方法: 重设 摄像机到 TargetTexture 的渲 染	38
2.2.7 ViewportPointToRay 方法: 近 视口到屏幕的射线	39
2.2.8 ViewportToWorldPoint 方法: 坐标点的坐标系转换	41
2.2.9 WorldToScreenPoint 方法: 坐 标点的坐标系转换	42
2.2.10 WorldToViewportPoint 方法: 坐标点的坐标系转换	44

2.3 关于 Camera 视口、aspect、pixelRect 及 rect 的关系注解	45
第 3 章 GameObject 类	49
3.1 GameObject 类实例属性	49
3.2 GameObject 构造方法	51
3.3 GameObject 类实例方法	52
3.3.1 GetComponent 方法: 获取组 件	52
3.3.2 SendMessage 方法: 发送消息	56
3.4 GameObject 类静态方法	58
3.5 关于 GameObject 类和 Component 类的 使用注解	60
第 4 章 HideFlags 类	62
4.1 HideFlags 类枚举成员	62
4.1.1 DontSave: 保留对象到新 场景	62
4.1.2 HideAndDontSave: 保留对象到 新场景	64
4.1.3 HideInHierarchy: 在 Hierarchy 面板中隐藏	65
4.1.4 HideInInspector: 在 Inspector 面板中隐藏	66
4.1.5 None: HideFlags 默认值	67
4.1.6 NotEditable: 对象在 Inspector 面板中的可编辑性	67
4.2 HideFlags 类使用小结	68
第 5 章 Mathf 类	69
5.1 Mathf 类静态属性	69
5.1.1 Deg2Rad 属性: 从角度到弧度 常量	69
5.1.2 Infinity 属性: 正无穷大	70
5.2 Mathf 类静态方法	71
5.2.1 Clamp 方法: 返回有限范围值	71
5.2.2 ClosestPowerOfTwo 方法: 返回 2 的某次幂	72
5.2.3 DeltaAngle 方法: 最小增量 角度	73
5.2.4 InverseLerp 方法: 计算比 例值	74
5.2.5 Lerp 方法: 线性插值	75
5.2.6 LerpAngle 方法: 角度插值	75
5.2.7 MoveTowards 方法: 选择性 插值	77
5.2.8 MoveTowardsAngle 方法: 角 度的选择性插值	78
5.2.9 PingPong 方法: 往复运动	79
5.2.10 Repeat 方法: 取模运算	80
5.2.11 Round 方法: 浮点数的整 型值	81
5.2.12 SmoothDamp 方法: 模拟阻 尼运动	83
5.2.13 SmoothDampAngle 方法: 阻尼 旋转	84
5.2.14 SmoothStep 方法: 平滑 插值	85
第 6 章 Matrix4x4 类	88
6.1 Matrix4x4 类实例方法	88
6.1.1 MultiplyPoint 方法: 投影矩 阵变换	88
6.1.2 MultiplyPoint3x4 方法: 矩阵 变换	89
6.1.3 MultiplyVector 方法: 矩阵 变换	89
6.1.4 SetTRS 方法: 重设 Matrix4x4 变换矩阵	91
6.2 Matrix4x4 类静态方法	92
6.2.1 Ortho 方法: 创建正交投影 矩阵	92
6.2.2 Perspective 方法: 创建透视 投影矩阵	93
6.2.3 TRS 方法: 返回 Matrix4x4 实例	95
第 7 章 Object 类	96
7.1 Object 类实例方法	96
7.2 Object 类静态方法	97
7.2.1 Destroy 方法: 销毁对象	97
7.2.2 DontDestroyOnLoad 方法: 新场景中保留对象	98

7.2.3	FindObjectsOfType 方法: 获取对象	100	9.1.1	insideUnitCircle 属性: 圆 内随机点	120
7.2.4	Instantiate 方法: 实例化 对象	101	9.1.2	rotationUniform 属性: 均匀 分布特征	121
第 8 章 Quaternion 类		102	9.1.3	seed 属性: 随机数种子	123
8.1	Quaternion 类实例属性	102	9.2	Random 类其他常用静态属性功能简 介	124
8.2	Quaternion 类实例方法	103	第 10 章 Rigidbody 类		125
8.2.1	SetFromToRotation 方法: 创 建 rotation 实例	103	10.1	Rigidbody 类实例属性	125
8.2.2	SetLookRotation 方法: 设置 Quaternion 实例的朝向	104	10.1.1	collisionDetectionMode 属 性: 碰撞检测模式	125
8.2.3	ToAngleAxis 方法: Quaternion 实例的角轴表示	106	10.1.2	drag 属性: 刚体阻力	127
8.3	Quaternion 类静态方法	107	10.1.3	inertiaTensor 属性: 惯性 张量	128
8.3.1	Angle 方法: Quaternion 实例 夹角	107	10.1.4	mass 属性: 刚体质量	129
8.3.2	Dot 方法: 点乘	108	10.1.5	velocity 属性: 刚体速度	131
8.3.3	Euler 方法: 欧拉角对应的 四元数	109	10.2	Rigidbody 类实例方法	132
8.3.4	FromToRotation 方法: Quaternion 变换	111	10.2.1	AddExplosionForce 方法: 模拟爆炸力	132
8.3.5	Inverse 方法: 逆向 Quaternion 值	112	10.2.2	AddForceAtPosition 方法: 增加刚体点作用力	135
8.3.6	Lerp 方法: 线性插值	113	10.2.3	AddTorque 方法: 刚体添加 扭矩	136
8.3.7	LookRotation 方法: 设置 Quaternion 的朝向	114	10.2.4	ClosestPointOnBounds 方法: 爆炸点到刚体最短距离	138
8.3.8	RotateTowards 方法: Quaternion 插值	115	10.2.5	GetPointVelocity 方法: 刚体点速度	139
8.3.9	Slerp 方法: 球面插值	116	10.2.6	GetRelativePointVelocity 方 法: 刚体点相对速度	140
8.4	Quaternion 类运算符	117	10.2.7	MovePosition 方法: 刚体位 置移动	142
8.4.1	operator*(lhs:Quaternion, rhs:Quaternion)	117	10.2.8	Sleep 方法: 刚体休眠	143
8.4.2	operator*(rotation:Quaternion, point:Vector3)	118	10.2.9	SweepTest 方法: 检测碰 撞器	144
8.5	关于 Quaternion 类中相乘运算符的 两种重载方式的注解	119	10.2.10	SweepTestAll 方法: 探测 碰撞器	146
第 9 章 Random 类		120	10.2.11	WakeUp 方法: 唤醒刚体	147
9.1	Random 类静态属性	120	10.3	关于 useGravity、isKinematic 和 velocity 的使用注解	147

10.4	关于 Rigidbody 中 mass、density 及 scale 之间的关系注解	150
10.5	关于作用力方式 ForceMode 的功能注解	151
10.6	关于 OnTriggerXXX 和 OnCollisionXXX 的功能注解	153
第 11 章	Time 类	158
11.1	Time 类静态属性	158
11.1.1	realtimeSinceStartup 属性: 程序运行实时时间	158
11.1.2	smoothDeltaTime 属性: 平滑时间间隔	159
11.1.3	time 属性: 程序运行时间	160
11.2	Time 类其他常用静态属性功能简介	162
第 12 章	Transform 类	163
12.1	Transform 类实例属性	163
12.1.1	eulerAngles 属性: 欧拉角	163
12.1.2	forward 属性: z 轴单位向量	164
12.1.3	hasChanged 属性: transform 组件是否被修改	165
12.1.4	localPosition 属性: 局部坐标系位置	166
12.1.5	localToWorldMatrix 属性: 转换矩阵	168
12.1.6	parent 属性: 父物体 Transform 实例	169
12.1.7	worldToLocalMatrix 属性: 转换矩阵	170
12.2	Transform 类实例方法	171
12.2.1	DetachChildren 方法: 分离物体层级关系	171
12.2.2	GetChild 方法: 获取 GameObject 对象子类	172
12.2.3	InverseTransformDirection 方法: 坐标系转换	173
12.2.4	InverseTransformPoint 方法: 点的相对坐标向量	174
12.2.5	IsChildOf 方法: 是否为子物体	175
12.2.6	LookAt 方法: 物体朝向	176
12.2.7	Rotate 方法: 绕坐标轴旋转	177
12.2.8	Rotate 方法: 绕某个向量旋转	178
12.2.9	RotateAround 方法: 绕轴点旋转	179
12.2.10	TransformDirection 方法: 坐标系转换	180
12.2.11	TransformPoint 方法: 点的世界坐标位置	181
12.2.12	Translate 方法: 相对坐标系移动	182
12.2.13	Translate 方法: 相对其他物体移动	183
12.3	关于 localScale 和 lossyScale 的功能注解	184
12.4	关于 Transform 类中涉及空间变换的几个属性和方法的功能注解	186
第 13 章	Vector2 类	190
13.1	Vector2 类实例方法	190
13.2	Vector2 类静态方法	191
13.2.1	Angle 方法: 两个向量夹角	191
13.2.2	ClampMagnitude 方法: 向量长度	192
13.2.3	Lerp 方法: 向量插值	193
13.2.4	MoveTowards 方法: 向量插值	194
13.2.5	Scale 方法: 向量放缩	196
13.3	Vector2 类运算符	197
第 14 章	Vector3 类	199
14.1	Vector3 类实例属性	199
14.1.1	normalized 属性: 单位化向量	199
14.1.2	sqrMagnitude 属性: 模长平方	200

14.2	Vector3 类实例方法	201	14.3.11	RotateTowards 方法: 球形插值	215
14.3	Vector3 类静态方法	203	14.3.12	Scale 方法: 向量放缩	216
14.3.1	Angle 方法: 求两个向量夹角	203	14.3.13	Slerp 方法: 球形插值	218
14.3.2	ClampMagnitude 方法: 向量长度	203	14.3.14	SmoothDamp 方法: 阻尼移动	219
14.3.3	Cross 方法: 向量叉乘	205	14.4	Vector3 类运算符	220
14.3.4	Dot 方法: 向量点乘	206	14.5	关于 Vector3.Lerp 和 Vector3.MoveTowards 的功能注解	221
14.3.5	Lerp 方法: 向量插值	207	14.6	关于 Vector3.RotateTowards 和 Vector3.Slerp 的功能注解	222
14.3.6	MoveTowards 方法: 向量插值	208	第 15 章 游戏实例——坚守阵地	223	
14.3.7	OrthoNormalize 方法: 两个坐标轴的正交化	209	15.1	游戏概述	223
14.3.8	OrthoNormalize 方法: 3 个坐标轴的正交化	211	15.2	建模与导入	225
14.3.9	Project 方法: 投影向量	213	15.3	程序脚本	229
14.3.10	Reflect 方法: 反射向量	214	15.4	制作简单小地图	242

Application类

Application类不含实例属性和实例方法，在脚本中通过直接调用Application类的静态属性和静态方法来控制程序的运行时数据，如场景的管理、数据的加载等。本章主要介绍Application类的一些静态属性和静态方法。

1.1 Application 类静态属性

在Application类中，涉及的静态属性主要有dataPath和loadedLevel。由于Application类的persistentDataPath属性、streamingAssetsPath属性和temporaryCachePath属性的功能与dataPath属性功能相近，因此把这些属性放到一起介绍。下面详细介绍这些属性。

1.1.1 dataPath属性：数据文件路径

基本语法 `public static string dataPath { get; }`

功能说明 此属性用于返回程序的数据文件所在文件夹的路径（只读）。返回路径为相对路径，不同游戏平台的数据文件保存路径不同，具体如下所示。

- Unity Editor: <工程文件夹所在路径>/Assets。
- Mac player: <应用程序路径>/Contents。
- iPhone player: <应用程序路径>/<AppName.app>/Data。
- Win player: <包含可执行文件的文件夹路径>/Data。
- Web player: 播放器数据文件夹的绝对路径（没有实际的数据文件名称）。
- Flash: 播放器数据文件夹的绝对路径（没有实际的数据文件名称）。

提 示 与此属性功能相近的属性有persistentDataPath、streamingAssetsPath和temporaryCachePath，它们的具体功能如下所示。

- persistentDataPath: 此属性用于返回一个持久化数据存储目录的路径（只读），可以在此路径下存储一些持久化的数据文件。对于同一平台，在不同程序中调用此属性时，其返回值是相同的，但是在不同的运行平台下，其返回值会不一样。

- `streamingAssetsPath`: 此属性用于返回流数据的缓存目录, 返回路径为相对路径, 适合设置一些外部数据文件的路径。
- `temporaryCachePath`: 此属性用于返回一个临时数据的缓存目录(只读)。对于同一平台, 在不同程序中调用此属性时, 其返回值是相同的, 但是在不同的运行平台下, 其返回值是不一样的。

实例演示 以下代码演示了4种不同路径的属性输出值:

```
using UnityEngine;
using System.Collections;

public class DataPath_ts : MonoBehaviour
{
    void Start()
    {
        //4种不同的路径, 都为只读
        //dataPath和streamingAssetsPath的路径位置一般是相对程序的安装目录位置
        //persistentDataPath和temporaryCachePath的路径位置一般是相对所在系统的固定位置
        Debug.Log("dataPath:" + Application.dataPath);
        Debug.Log("persistentDataPath:" + Application.persistentDataPath);
        Debug.Log("streamingAssetsPath:" + Application.streamingAssetsPath);
        Debug.Log("temporaryCachePath:" + Application.temporaryCachePath);
    }
}
```

在这段代码中, 分别打印出了4种不同的路径模式, 其中`dataPath`和`streamingAssetsPath`这两个属性的返回值一般是相对于程序安装目录的位置。由于是相对路径, 这两个属性非常适用于在多平台移植中设置要读取的外部数据文件的路径。在第15章的综合实例中, 就用到了`dataPath`这个属性。而`persistentDataPath`和`temporaryCachePath`这两个属性的返回值一般是程序所在平台的固定位置, 对于不同的平台, 其位置是不一样的, 适合存放程序运行过程中产生的一些数据文件。图1-1是程序在我电脑中的运行输出, 从输出结果可以看出这4种属性的不同路径返回值。

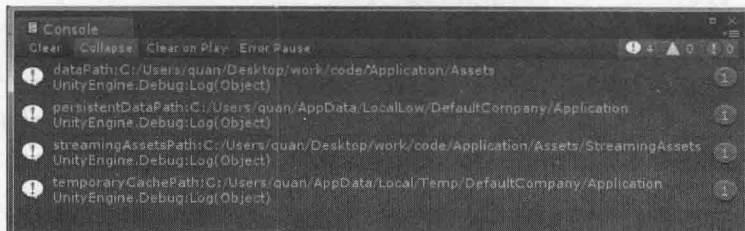


图1-1 `dataPath`实例演示的运行结果

1.1.2 loadedLevel属性: 关卡索引

基本语法 `public static int loadedLevel { get; }`

功能说明 此属性用于返回当前程序最后加载的关卡（即Scene）的索引值（只读）。

实例演示 下面通过实例演示loadedLevel属性的使用方法。

```
using UnityEngine;
using System.Collections;
public class LoadedLevel_ts : MonoBehaviour
{
    void Start()
    {
        //返回当前场景的索引值
        Debug.Log("loadedLevel:" + Application.loadedLevel);
        //返回当前场景的名字
        Debug.Log("loadedLevelName:" + Application.loadedLevelName);
        //是否有场景正在被加载
        //在使用Application类的静态方法LoadLevel或LoadLevelAdditive加载一个新的场景时,
        //常常需要持续一段时间才能加载完毕,当场景加载完毕时,isLoadingLevel返回true,
        //否则返回false
        Debug.Log("isLoadingLevel:" + Application.isLoadingLevel);
        //返回游戏中可被加载的场景数量
        Debug.Log("levelCount:" + Application.levelCount);
        //返回当前游戏的运行平台
        //游戏的运行平台有很多种,例如手机、电脑、游戏机等,具体类型可在枚举类
        //RuntimePlatform中查看
        Debug.Log("platform:" + Application.platform);
        //当前游戏是否正在运行
        Debug.Log("isPlaying:" + Application.isPlaying);
        //当前游戏是否处于Unity编辑模式
        Debug.Log("isEditor:" + Application.isEditor);
    }
}
```

在这段代码中，分别打印出了Application类的一些常用静态属性，具体的属性功能请参考代码中的注释。图1-2是代码执行的输出结果，其中levelCount的值为2，这是因为在BuildSettings（File→BuildSettings）中有两个场景，具体请到工程中查看。

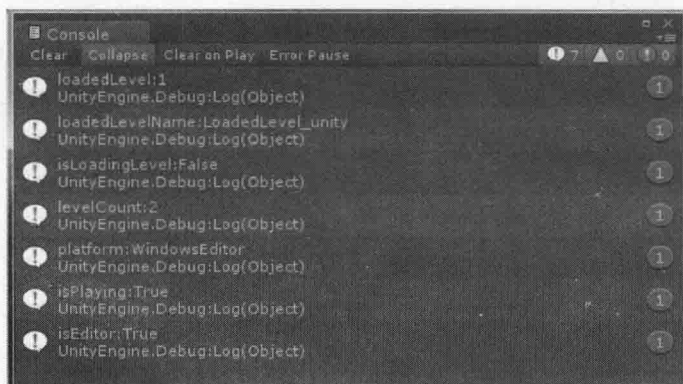


图1-2 属性loadedLevel的实例演示的运行结果

1.2 Application 类静态方法

在Application类中，涉及的静态方法有CaptureScreenshot方法、LoadLevelAdditiveAsync方法和RegisterLogCallback方法，下面详细介绍这些方法。

1.2.1 CaptureScreenshot方法：截屏

基本语法 (1) public static void CaptureScreenshot(string filename);

(2) public static void CaptureScreenshot(string filename, int superSize);

其中参数filename为截屏文件名称，superSize为放大系数，默认为0，即不放大。

功能说明 此方法用于截取当前游戏画面并将截取的图片保存为PNG格式。截屏后文件会默认保存在根目录下，如果根目录下已存在同名文件，将会被替换。当superSize大于1时，截屏文件的宽度和高度将同时被放大superSize倍。

提 示

- 此方法在 Web 模式下无效。
- 当放大系数小于0时，按默认值0处理，即图片不放大也不缩小。

实例演示 下面通过实例演示如何使用CaptureScreenshot方法来截屏。

```
using UnityEngine;
using System.Collections;

public class CaptureScreenshot_ts : MonoBehaviour
{
    int tp = -1;
    void Update()
    {
        if (tp == 0)
        {
            //默认值，不放大
            Application.CaptureScreenshot("test01.png", 0);
        }
        else if (tp == 1)
        {
            //放大系数为1，即不放大
            Application.CaptureScreenshot("test02.png", 1);
        }
        else
        {
            //放大系数为2，即放大2倍
            Application.CaptureScreenshot("test03.png", 2);
        }
        tp++;
    }
}
```

在这段代码中，首先声明了一个int类型的变量tp，然后在Update方法中根据不同的变量值生成了3张不同放大系数的PNG图片。请自行运行程序，在程序根目录下查看生成的PNG文件的大小。由于CaptureScreenshot方法在每帧中只执行一次，所以如果使用如下代码的话，只会生成一张PNG图片，即test03.png：

```
using UnityEngine;
using System.Collections;

public class CaptureScreenshot_ts : MonoBehaviour
{
    void Update() {
        Application.CaptureScreenshot("test01.png", 0);
        Application.CaptureScreenshot("test02.png", 1);
        Application.CaptureScreenshot("test03.png", 2);
    }
}
```

在Update方法中，依次调用了3次CaptureScreenshot方法，试图生成3个不同放大系统的图片，但是在同一帧中，只有最后一次调用才能生效，故此段程序的运行结果只能生成一张PNG图片，即test03.png。

由于CaptureScreenshot方法可以实时截取程序屏幕，因此可以用其截图所包含的信息做一些有趣的应用，下面是一个小例子。

在本例中，玩家可以在程序左上角的TextField输入框里输入几个文字（最多不要超过输入框的最大宽度），单击“确定”按钮便会看到很多小球组成了文字的形状分布在三维空间中。图1-3是程序启动后的截图，图1-4是单击“确定”后的截图，图1-5是在文本框中输入“霞光满天”四个字后显示的形状。

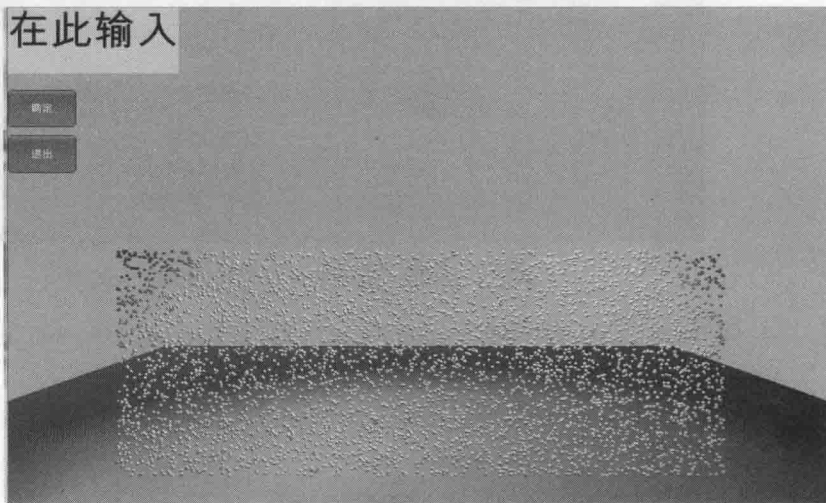


图1-3 程序启动后界面

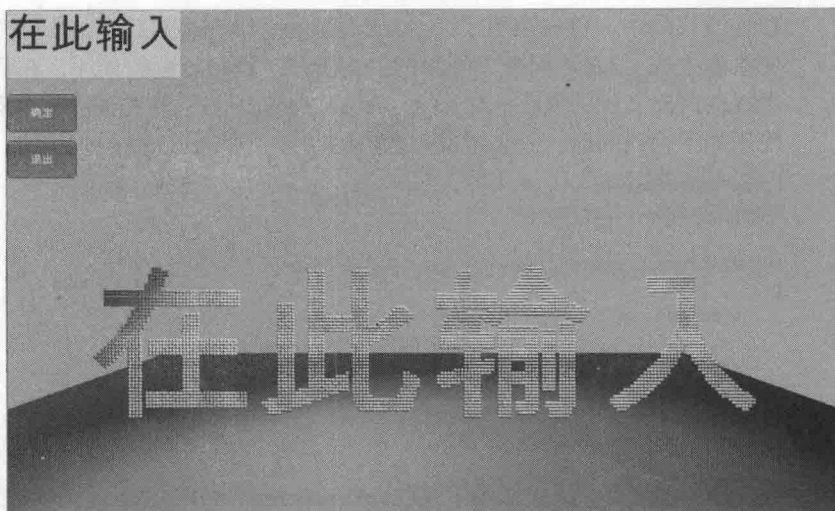


图1-4 单击“确定”按钮后的界面

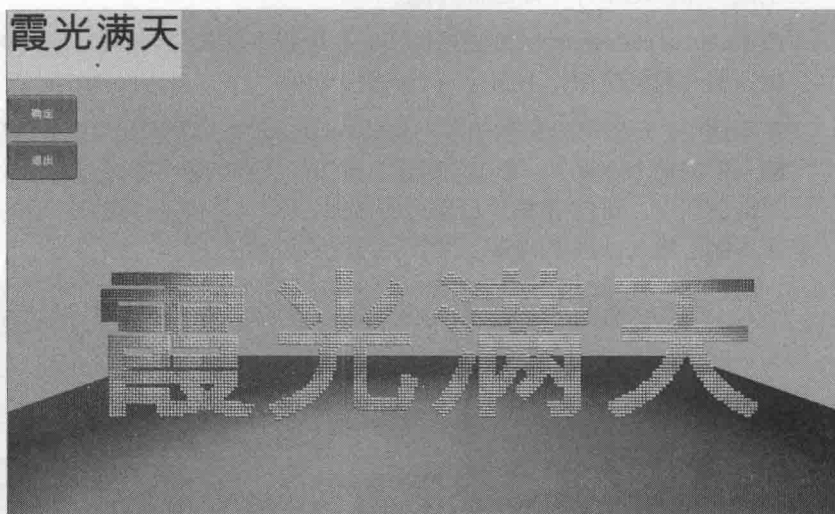


图1-5 在输入框中输入“霞光满天”后的效果

本例中用到的脚本有 `Capture_Use_ts.cs` 和 `Capture_Use_Sub_ts.cs`，其中脚本 `Capture_Use_Sub_ts.cs` 用来控制每个小球的位置，其代码如下。

```
using UnityEngine;
using System.Collections;

public class Capture_Use_Sub_ts : MonoBehaviour
{
    //物体移动的目标位置
```