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

Unity API 解析

陈泉宏◎编著

人民邮电出版社
北京

图书在版编目（CIP）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　　2014年9月第1版
印数：1－3 000册　　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 是个入门快、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状况。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

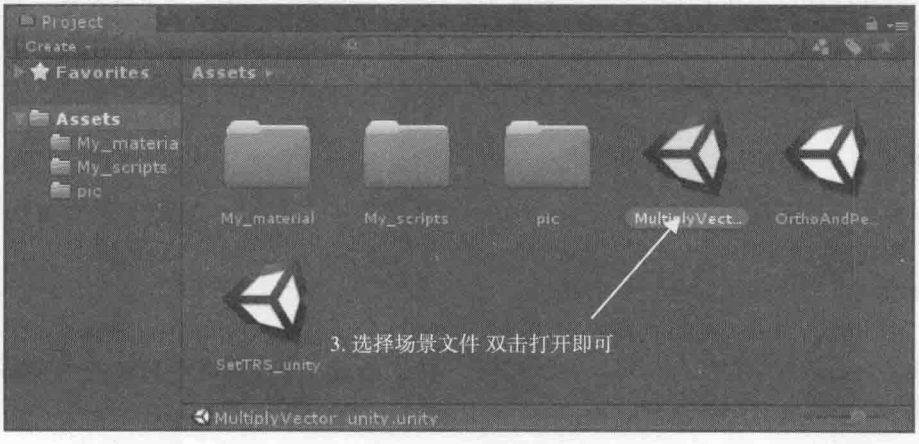
在本书中，以下专业用语的意义是相同的：“变量”与“属性”，“函数”与“方法”，“归一化”与“单位化”。

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

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。

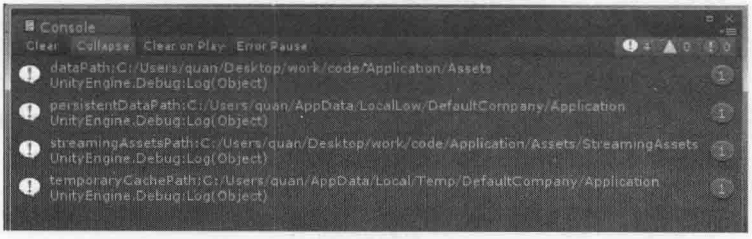


图1-1　dataPath实例演示的运行结果

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。

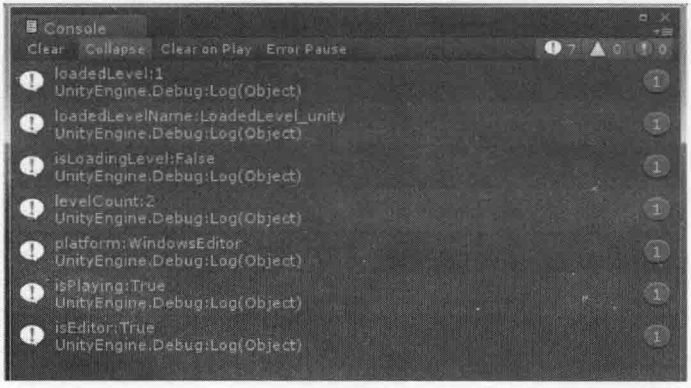


图1-2 属性loadedLevel的实例演示的运行结果

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。

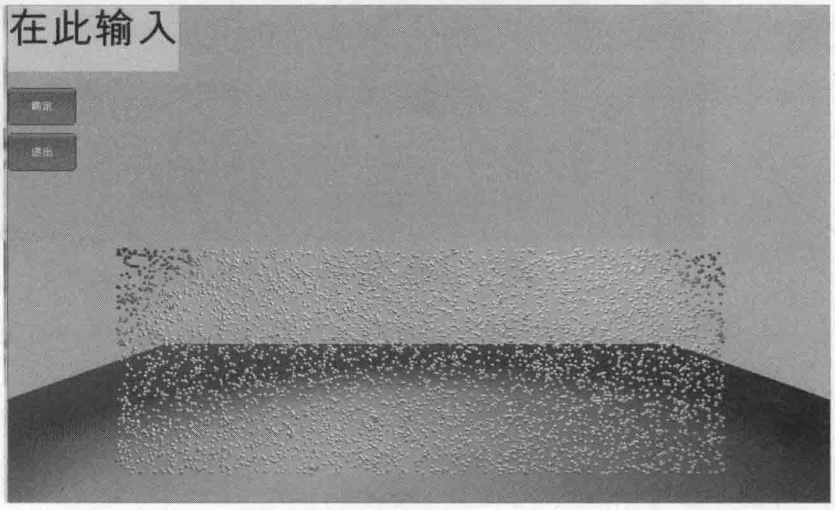


图1-3 程序启动后界面

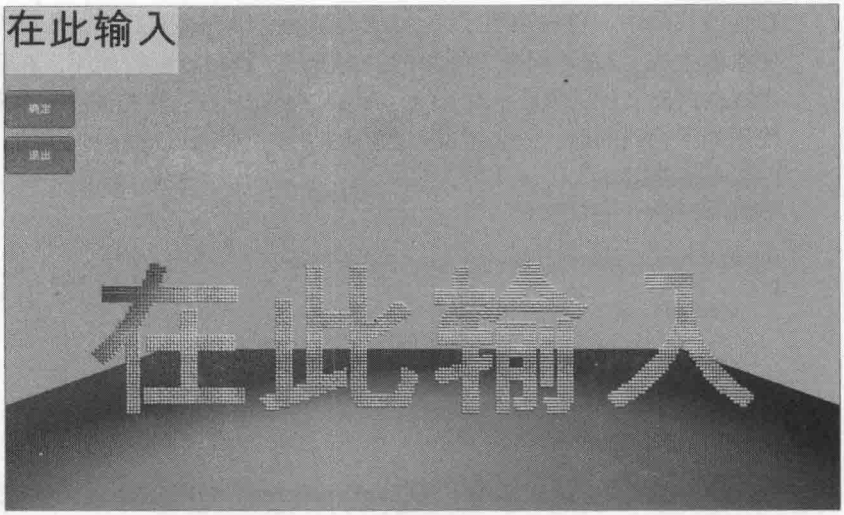


图1-4　单击“确定”按钮后的界面

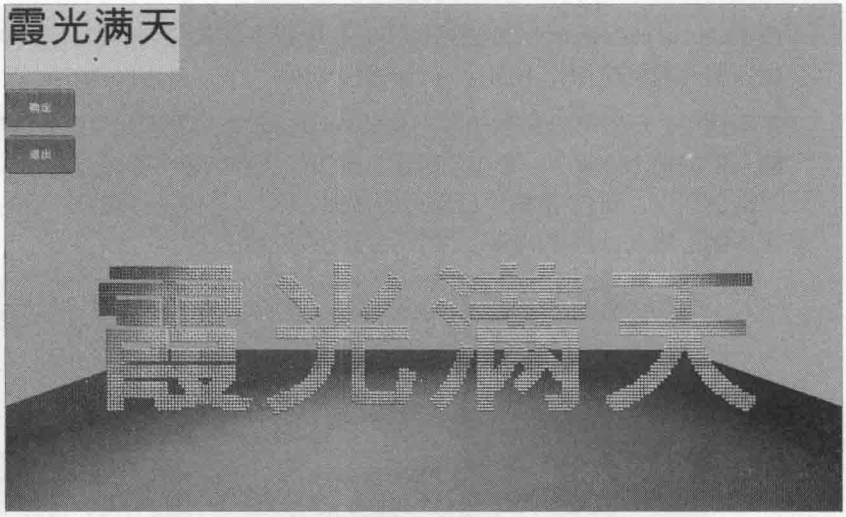


图1-5　在输入框中输入“霞光满天”后的效果

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
```