**Unity3D教程：Notification Center的简单使用与案例说明**

Posted on 2013年06月14日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 71 次

Notification Center的脚本源码如下：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 001 |  |
| 002 | import System.Collections.Generic; |
| 003 |  |
| 004 | *// NotificationCenter is used for handling messages between GameObjects.* |
| 005 |  |
| 006 | *// GameObjects can register to receive specific notifications. When another objects sends a notification of that type, all GameObjects that registered for it and implement the appropriate message will receive that notification.* |
| 007 |  |
| 008 | *// Observing GameObjetcs must register to receive notifications with the AddObserver function, and pass their selves, and the name of the notification. Observing GameObjects can also unregister themselves with the RemoveObserver function. GameObjects must request to receive and remove notification types on a type by type basis.* |
| 009 |  |
| 010 | *// Posting notifications is done by creating a Notification object and passing it to PostNotification. All receiving GameObjects will accept that Notification object. The Notification object contains the sender, the notification type name, and an option hashtable containing data.* |
| 011 |  |
| 012 | *// To use NotificationCenter, either create and manage a unique instance of it somewhere, or use the static NotificationCenter.* |
| 013 |  |
| 014 | *// We need a static method for objects to be able to obtain the default notification center.* |
| 015 | *// This default center is what all objects will use for most notifications. We can of course create our own separate instances of NotificationCenter, but this is the static one used by all.* |
| 016 | **private** **static** **var** defaultCenter : NotificationCenter; |
| 017 | **static** function DefaultCenter () { |
| 018 | *// If the defaultCenter doesn't already exist, we need to create it* |
| 019 | **if** (!defaultCenter) { |
| 020 | *// Because the NotificationCenter is a component, we have to create a GameObject to attach it to.* |
| 021 | **var** notificationObject: GameObject = new GameObject("Default Notification Center"); |
| 022 | *// Add the NotificationCenter component, and set it as the defaultCenter* |
| 023 | defaultCenter = notificationObject.AddComponent(NotificationCenter); |
| 024 | } |
| 025 |  |
| 026 | **return** defaultCenter; |
| 027 | } |
| 028 |  |
| 029 | *// Our hashtable containing all the notifications. Each notification in the hash table is an ArrayList that contains all the observers for that notification.* |
| 030 | **var** notifications: Hashtable = new Hashtable(); |
| 031 |  |
| 032 | *// AddObserver includes a version where the observer can request to only receive notifications from a specific object. We haven't implemented that yet, so the sender value is ignored for now.* |
| 033 | function AddObserver (observer, name: **String**) { AddObserver(observer, name, **null**); } |
| 034 | function AddObserver (observer, name: **String**, sender) { |
| 035 | *// If the name isn't good, then throw an error and return.* |
| 036 | **if** (name == **null** || name == "") { Debug.Log("Null name specified for notification in AddObserver."); **return**; } |
| 037 | *// If this specific notification doens't exist yet, then create it.* |
| 038 | **if** (!notifications[name]) { |
| 039 | notifications[name] = new List.<component>(); |
| 040 | }</component> |
| 041 |  |
| 042 | **var** notifyList: List.<component> = notifications[name];</component> |
| 043 |  |
| 044 | *// If the list of observers doesn't already contains the one that's registering, then add it.* |
| 045 | **if** (!notifyList.Contains(observer)) { notifyList.**Add**(observer); } |
| 046 | } |
| 047 |  |
| 048 | *// RemoveObserver removes the observer from the notification list for the specified notification type* |
| 049 | function RemoveObserver (observer, name: **String**) { |
| 050 | **var** notifyList: List.<component> = notifications[name];</component> |
| 051 |  |
| 052 | *// Assuming that this is a valid notification type, remove the observer from the list.* |
| 053 | *// If the list of observers is now empty, then remove that notification type from the notifications hash. This is for housekeeping purposes.* |
| 054 | **if** (notifyList) { |
| 055 | **if** (notifyList.Contains(observer)) { notifyList.**Remove**(observer); } |
| 056 | **if** (notifyList.Count == 0) { notifications.**Remove**(name); } |
| 057 | } |
| 058 | } |
| 059 |  |
| 060 | *// PostNotification sends a notification object to all objects that have requested to receive this type of notification.* |
| 061 | *// A notification can either be posted with a notification object or by just sending the individual components.* |
| 062 | function PostNotification (aSender, aName: **String**) { PostNotification(aSender, aName, **null**); } |
| 063 | function PostNotification (aSender, aName: **String**, aData) { PostNotification(new Notification(aSender, aName, aData)); } |
| 064 | function PostNotification (aNotification: Notification) { |
| 065 | *// First make sure that the name of the notification is valid.* |
| 066 | *//Debug.Log("sender: " + aNotification.name);* |
| 067 | **if** (aNotification.name == **null** || aNotification.name == "") { Debug.Log("Null name sent to PostNotification."); **return**; } |
| 068 | *// Obtain the notification list, and make sure that it is valid as well* |
| 069 | **var** notifyList: List.<component> = notifications[aNotification.name]; |
| 070 | **if** (!notifyList) { Debug.Log("Notify list not found in PostNotification: " + aNotification.name); **return**; }</component> |
| 071 |  |
| 072 | *// Create an array to keep track of invalid observers that we need to remove* |
| 073 | **var** observersToRemove = new List.<component>();</component> |
| 074 |  |
| 075 | *// Itterate through all the objects that have signed up to be notified by this type of notification.* |
| 076 | **for** (**var** observer **in** notifyList) { |
| 077 | *// If the observer isn't valid, then keep track of it so we can remove it later.* |
| 078 | *// We can't remove it right now, or it will mess the for loop up.* |
| 079 | **if** (!observer) { observersToRemove.**Add**(observer); |
| 080 | } **else** { |
| 081 | *// If the observer is valid, then send it the notification. The message that's sent is the name of the notification.* |
| 082 | observer.SendMessage(aNotification.name, aNotification, SendMessageOptions.DontRequireReceiver); |
| 083 | } |
| 084 | } |
| 085 |  |
| 086 | *// Remove all the invalid observers* |
| 087 | **for** (observer **in** observersToRemove) { |
| 088 | notifyList.**Remove**(observer); |
| 089 | } |
| 090 | } |
| 091 |  |
| 092 | *// The Notification class is the object that is send to receiving objects of a notification type.* |
| 093 | *// This class contains the sending GameObject, the name of the notification, and optionally a hashtable containing data.* |
| 094 | **class** Notification { |
| 095 | **var** sender; |
| 096 | **var** name : **String**; |
| 097 | **var** data; |
| 098 | function Notification (aSender, aName: **String**) { sender = aSender; name = aName; data = **null**; } |
| 099 | function Notification (aSender, aName: **String**, aData) { sender = aSender; name = aName; data = aData; } |
| 100 | } |

接下来就是在需要的地方声明这个通知。

|  |  |  |
| --- | --- | --- |
|  |  |  |

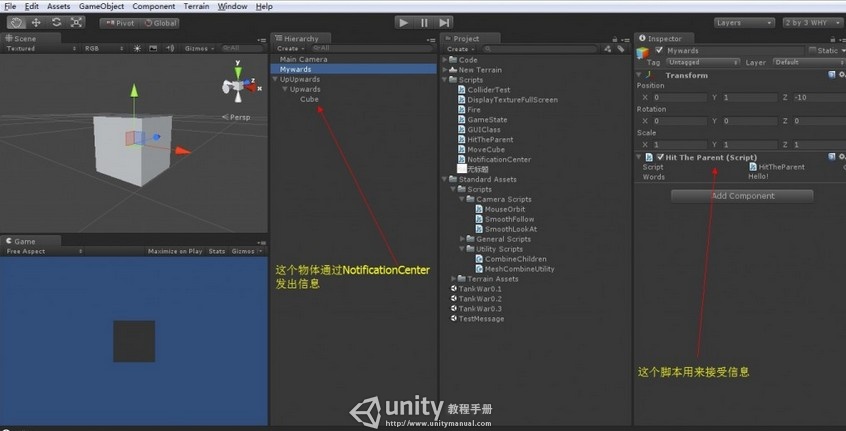
|  |  |
| --- | --- |
| 01 | function Update () { |
| 02 | **if**(Input.GetButtonDown("Fire1")){ |
| 03 | **var** ray = Camera.main.ScreenPointToRay(Input.mousePosition); |
| 04 | **var** hit : RaycastHit; |
| 05 | **if**(Physics.Raycast(ray,hit)){ |
| 06 | *//hit.collider.BroadcastMessage("ApplyDamage",1,SendMessageOptions.DontRequireReceiver);* |
| 07 | NotificationCenter.DefaultCenter().PostNotification(**this**,"ApplyDamage"); |
| 08 | } |
| 09 | } |
| 10 | } |
| 11 |  |

有了发送通知的事件，还需要接收通知的GameObject，在接受体的脚本中写下以下源码：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | #pragma strict |
| 2 | **var** words:**String**; |
| 3 | function Start(){ |
| 4 | NotificationCenter.DefaultCenter().AddObserver(**this**,"ApplyDamage"); |
| 5 | } |
| 6 | function ApplyDamage(){ |
| 7 | Debug.Log(words); |
| 8 | } |

这样就能实现任意GameObject之间的通知传输和函数调用了。

[](http://www.unitymanual.com/wp-content/uploads/2013/06/QQ截图20130614142516.jpg)

Unity3D教程：Notification Center的简单使用与案例说明

当然，NotificationCenterCenter的作用远不止这么简单，它也可以用作数据传输，

查看一下官网的声明：

[](http://www.unitymanual.com/wp-content/uploads/2013/06/63.jpg)

Unity3D教程：Notification Center的简单使用与案例说明

可见还有一个重载的函数可以传参aData。使用起来也很方便，将脚本稍作修改即可：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | function Start(){ |
| 2 | NotificationCenter.DefaultCenter().AddObserver(**this**,"ApplyDamage"); |
| 3 | } <!--DVFMTSC--> *//Unity3D教程手册* |
| 4 | function ApplyDamage(note:Notification){ |
| 5 | Debug.Log("从"+note.sender+"接收一个信息内容"+note.data+"通知名称为"+note.name); |
| 6 | } |

运行效果便是：

[](http://www.unitymanual.com/wp-content/uploads/2013/06/27.jpg)

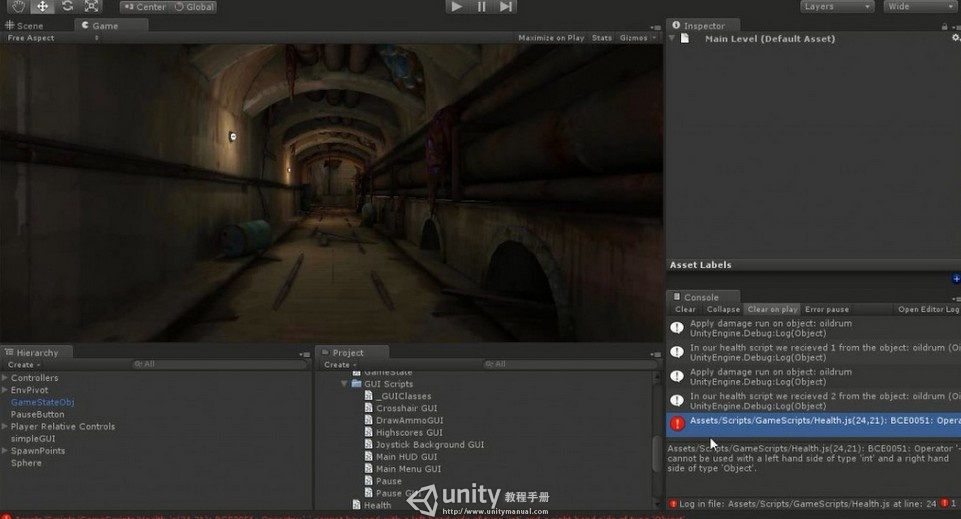
Unity3D教程：Notification Center的简单使用与案例说明

当然很多时候我们并不仅仅需要将数据打印出来，所以数据的转换也是一个非常关键的步骤。比如说下面这个案例：

[](http://www.unitymanual.com/wp-content/uploads/2013/06/35.jpg)

Unity3D教程：Notification Center的简单使用与案例说明

我们要输出剩余的Health值便会发现存在数据类型转换的问题：

[](http://www.unitymanual.com/wp-content/uploads/2013/06/43.jpg)

Unity3D教程：Notification Center的简单使用与案例说明

此时我们需要进行转换：

[](http://www.unitymanual.com/wp-content/uploads/2013/06/53.jpg)

Unity3D教程：Notification Center的简单使用与案例说明

这样就解决数据传输中的类型转换问题了。