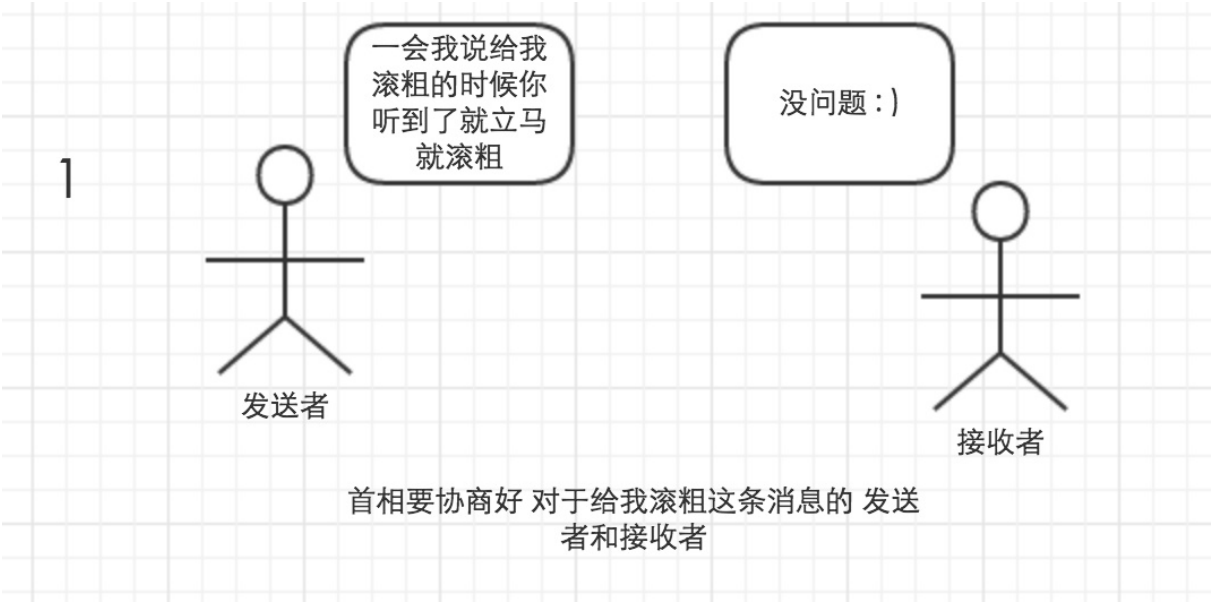
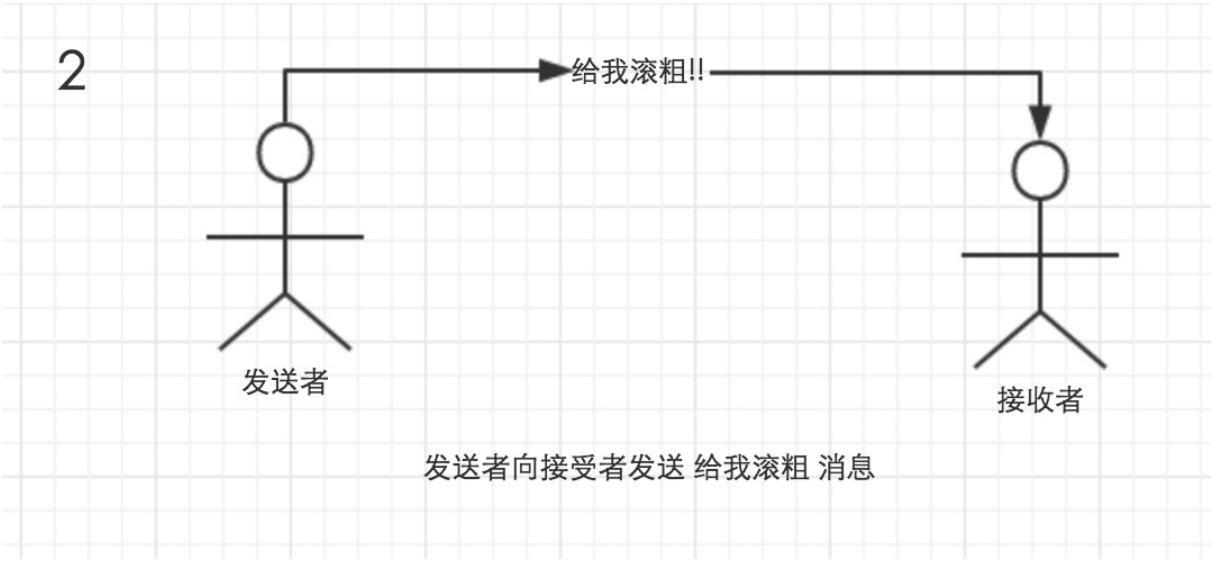
# Unity 游戏框架搭建
# 2017（五）简易消息机制

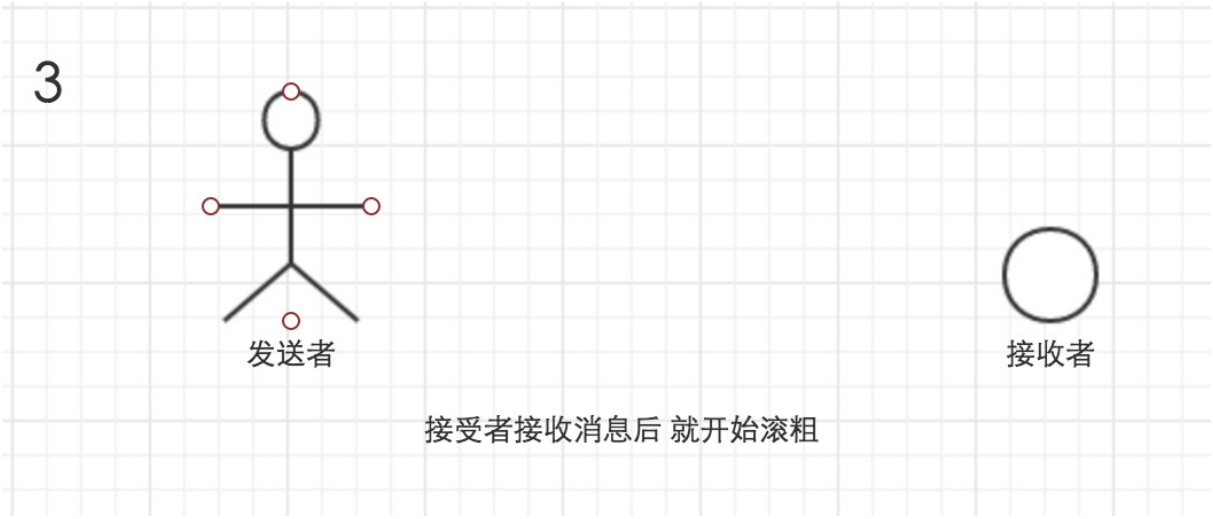
## 什么是消息机制?



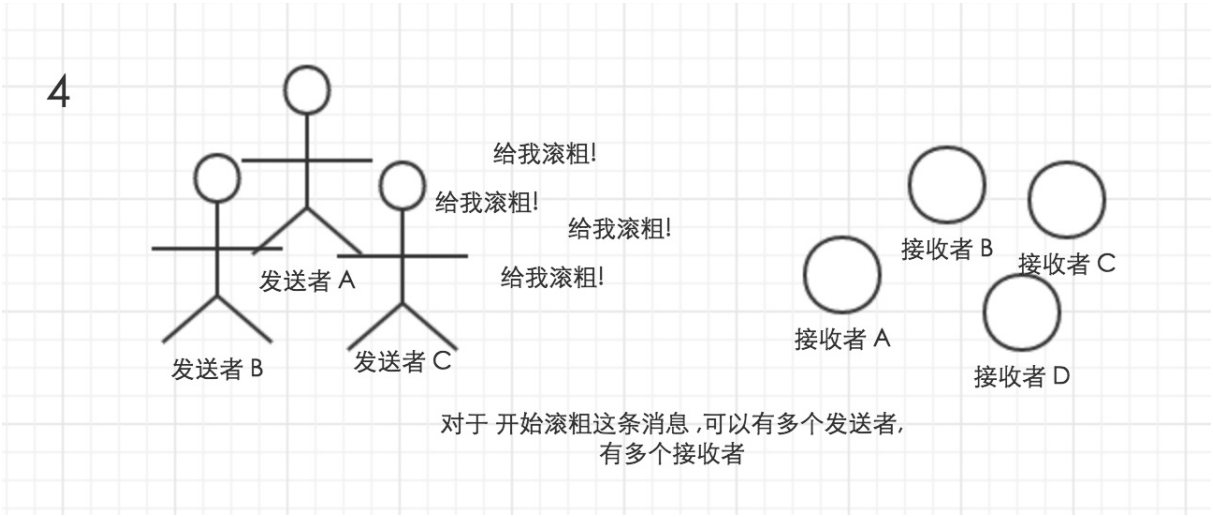

*DraggedImage.png*



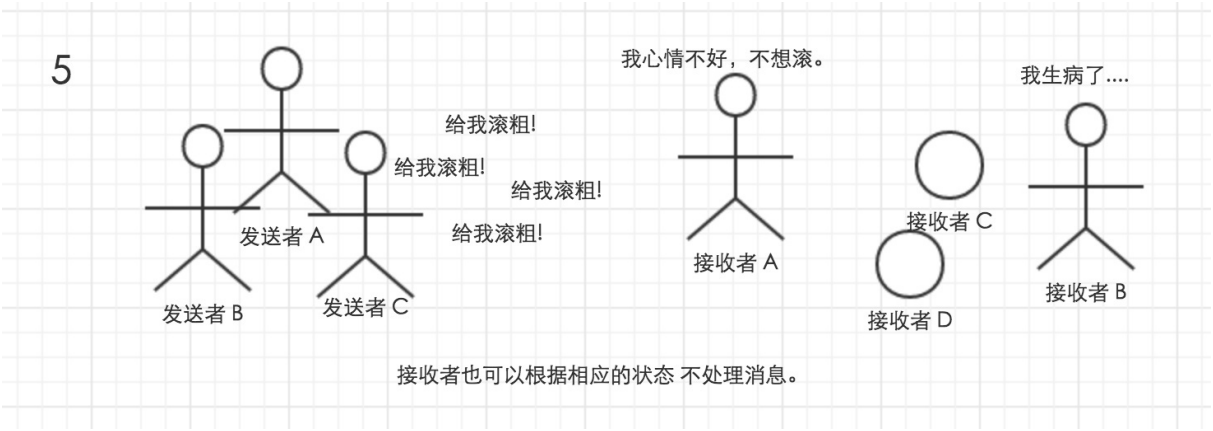

*DraggedImage-1.png*




*DraggedImage-2.png*




*DraggedImage-3.png*




*DraggedImage-4.png*

23333333，让我先笑一会。

# 为什么用消息机制?

三个字,解!!!!耦!!!!合!!!!。

# 我的框架中的消息机制用例:

1.接收者

Receiver.cs

```
using UnityEngine;

namespace QFramework.Example
{
    /// <summary>
    /// 教程地址:http://liangxiegame.com/post/5/
    /// </summary>
    public class Receiver : MonoBehaviour, IMsgReceiver
    {
        // Use this for initialization
        private void Awake()
        {
            this.RegisterLogicMsg("Receiver Show Sth",
ReceiveMsg);
        }

        private void ReceiveMsg(object[] args)
        {
            foreach (var arg in args)
            {
                Log.I(arg);
            }
        }
    }
}
```

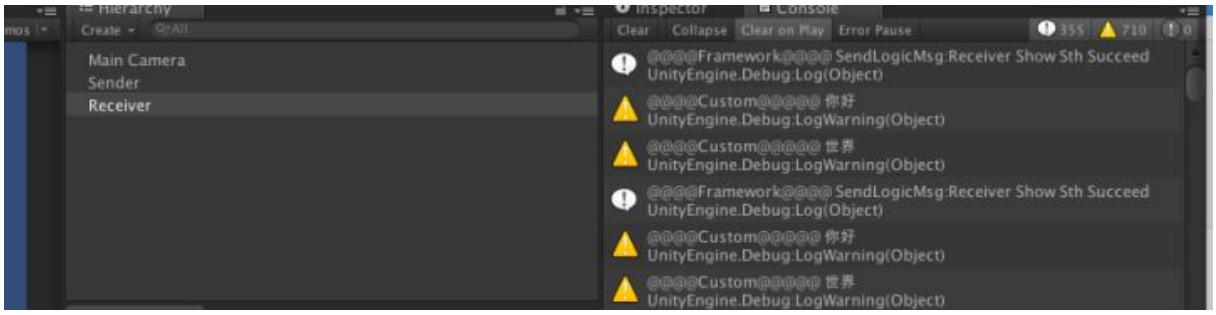
2.发送者

```
using UnityEngine;

namespace QFramework.Example
{
    /// <summary>
    /// 教程地址:http://liangxiegame.com/post/5/
    /// </summary>
```

```
    public class Sender : MonoBehaviour, IMsgSender
    {
        void Update()
        {
            this.SendLogicMsg("Receiver Show Sth", "你好", "世
界");
        }
    }
}
```

3.运行结果




*v2-4de41c395a2b7db148bee4cec926411e_hd.jpg*

使用起来几行代码的事情,实现起来就没这么简单了。

# 如何实现的?

可以看到接收者实现了接口IMsgReceiver,发送者实现了接口 IMsgSender。

那先看下这两个接口定义。

IMsgReceiver.cs:

```
namespace QFramework
{
    public interface IMsgReceiver
    {
    }
}
```

IMsgSender.cs:

```
namespace QFramework
{
    public interface IMsgSender
    {
    }
}
```

毛都没有啊。也没有 SendLogicMsg 或者 ReceiveLogicMsg 方法的定义啊。

答案是使用 C# this 的扩展方式实现接口方法。

不清楚的童鞋请百度 C# this 扩展,有好多文章就不介绍了。

以上先告一段落,先介绍个重要的角色,MsgDispatcher (消息分发器)。

贴上第一部分代码:

```
    /// <summary>
    /// 消息分发器
    /// C# this扩展 需要静态类
    /// 教程地址:http://liangxiegame.com/post/5/
    public static class MsgDispatcher
    {
        /// <summary>
        /// 消息捕捉器
        /// </summary>
        private class LogicMsgHandler
        {
            public readonly IMsgReceiver Receiver;
            public readonly Action<object[]> Callback;

            public LogicMsgHandler(IMsgReceiver receiver,
Action<object[]> callback)
            {
                Receiver = receiver;
                Callback = callback;
            }
        }

        /// <summary>
        /// 每个消息名字维护一组消息捕捉器。
```

```
        /// </summary>
        static readonly Dictionary<string, List<LogicMsgHandler>>
mMsgHandlerDict =
            new Dictionary<string, List<LogicMsgHandler>>();
```

读注释!!!

贴上注册消息的代码

```
        /// <summary>
        /// 注册消息,
        /// 注意第一个参数,使用了C# this的扩展,
        /// 所以只有实现IMsgReceiver的对象才能调用此方法
        /// </summary>
        public static void RegisterLogicMsg(this IMsgReceiver
self, string msgName, Action<object[]> callback)
        {
            // 略过
            if (string.IsNullOrEmpty(msgName))
            {
                Log.W("RegisterMsg:" + msgName + " is Null or
Empty");
                return;
            }

            // 略过
            if (null == callback)
            {
                Log.W("RegisterMsg:" + msgName + " callback is
Null");
                return;
            }

            // 略过
            if (!mMsgHandlerDict.ContainsKey(msgName))
            {
                mMsgHandlerDict[msgName] = new
List<LogicMsgHandler>();
            }

            // 看下这里
            var handlers = mMsgHandlerDict[msgName];
```

```
            // 略过
            // 防止重复注册
            foreach (var handler in handlers)
            {
                if (handler.Receiver == self && handler.Callback
== callback)
                {
                    Log.W("RegisterMsg:" + msgName + " ayready
Register");
                    return;
                }
            }

            // 再看下这里
            handlers.Add(new LogicMsgHandler(self, callback));
        }
```

为了节省您时间,略过部分的代码就不要看了,什么?!!你都看了!!!! 23333

发送消息相关的代码

```
        /// <summary>
        /// 发送消息
        /// 注意第一个参数
        /// </summary>
        public static void SendLogicMsg(this IMsgSender sender,
string msgName, params object[] paramList)
        {
            // 略过,不用看
            if (string.IsNullOrEmpty(msgName))
            {
                Log.E("SendMsg is Null or Empty");
                return;
            }

            // 略过,不用看
            if (!mMsgHandlerDict.ContainsKey(msgName))
            {
                Log.W("SendMsg is UnRegister");
                return;
```

```
            }

            // 开始看!!!!
            var handlers = mMsgHandlerDict[msgName];
            var handlerCount = handlers.Count;

            // 之所以是从后向前遍历,是因为  从前向后遍历删除后索引值会不断
变化
            // 参考文章,http://www.2cto.com/kf/201312/266723.html
            for (var index = handlerCount - 1; index >= 0;
index--)
            {
                var handler = handlers[index];

                if (handler.Receiver != null)
                {
                    Log.W("SendLogicMsg:" + msgName + "
Succeed");
                    handler.Callback(paramList);
                }
                else
                {
                    handlers.Remove(handler);
                }
            }
        }
```

OK 主要的部分全都贴出来啦。

# 可以改进的地方:

- 目前整个游戏的消息都由一个字典维护,可以改进为每个模块维护一个字典或者其他方式。
- 消息名字类型由字符串定义的,可以改成枚举转 unsigned int 方式。
- 欢迎补充。

# 坑:

- 如果是 MonoBehaviour 注册消息之后,GameObject Destroy 之前一定要注销消息,之前的解决方案是,自定义一个基类来维护该对象已经注册的消息列表,然后在基类的 OnDestory 时候遍历卸载。
- 欢迎补充。

转载请注明地址：凉鞋的笔记：liangxiegame.com

# 更多内容

- QFramework 地址：https://github.com/liangxiegame/QFramework
- QQ 交流群：623597263
- **Unity 进阶小班**：
  - 主要训练内容：
    - 框架搭建训练（第一年）
    - 跟着案例学 Shader（第一年）
    - 副业的孵化（第二年、第三年）
  - 权益、授课形式等具体详情请查看《小班产品手册》：https://liangxiegame.com/master/intro
- 关注公众号：liangxiegame 获取第一时间更新通知及更多的免费内容。