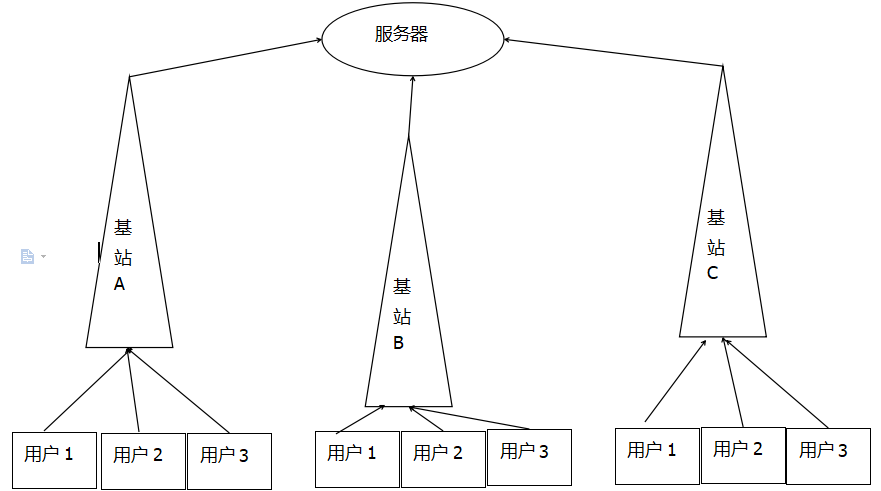
# Unity3D开发之我认为的万能应用框架

问了一早上群里的人帮我分析下这个框架的可行性，但是好像积极性都不是很高。挺失望的。算了，我就发在博客上，看到的愿意留下你们的建议，多谢。

由于我们部门经理还有一个同事都是做技术美工的，最近有点迷恋上unity3d开发，一直主动要求参加项目开发。之前一直是自己开发，随心所欲的 写，可发现代码的耦合性太高，根本没法放置一个模块给他们。上周开始一直再琢磨用一个什么框架能把模块分开清晰并且开发互不干扰。周六早上洗头的时候，突然就冒出了一个想法。因为我是移动用户，而移动通信的模式是经过千锤百炼的，所以我决定以中国移动通信的方式开发一套适合自己需求的框架。下面是中国移动信号传输的架构：



用户先发送消息给自己附近的基站，然后基站联系服务器，服务器解析所要拨打的电话号码所在的区域，然后发送电话指令给被叫用户附近的基站。基站接到指令后转送指令给被叫用户，一套完整的信息传输框架就出来了。我们可以类似移动的消息框架来设计我们的消息发送框架。

首先，我们给我们的项目划分模块。如：UI,Camera,Audio,Model,Data等等。这样我们分别创建我们对应模块的管理类来管理对应模块下的脚本。UImanager，CameraManager等等类似基站的功能，我们同时需要创建MessageCenter类来解析消息。MessageCenter相当于上图的服务器。各个模块的脚本就相当于用户，所有脚本之间的交互就是基于消息系统，这样各个脚本开发独立，互不影响。下面就是我们的代码实现。

首先我们创建一个BaseManager并让各个模块的Manager类继承自它，这样方便消息中心去管理Manager类。然后在BaseManager类里我们设置一个枚举类型 ，代码如下：

public enum ManagerCode

{

AUDIO=0,

CAMERA=5000,

DATA=10000,

MODEL=15000,

UI=20000

}

因为一个软件系统我们一般最多有七八个模块，封顶也就十分（偶尔还是会有多的，但都是极少数的）。而每个模块一般发送的消息1000条就已经很多了，所以，我们给每个模块5000条消息id值。

然后创建消息处理中心MessageCenter类，声明各个模块的管理类，并创建字典管理他们。字典的键值就是ManagerCode。代码如下：

//管理manager模块的字典

private Dictionary<ManagerCode, BaseManager> managerDic=new Dictionary<ManagerCode, BaseManager>();

我们在Awake函数里声明各个Manager并添加到字典中管理。记住要设置MessageCenter的执行顺序为最先。

void Awake()

{

\_Instance = this;

audioManager = new AudioManager(this);

cameraManager = new CameraManager(this);

dataManager = new DataManager(this);

modelManager = new ModelManager(this);

uiManager = new UIManager(this);

managerDic.Add(ManagerCode.AUDIO, audioManager);

managerDic.Add(ManagerCode.CAMERA, cameraManager);

managerDic.Add(ManagerCode.DATA, dataManager);

managerDic.Add(ManagerCode.MODEL, modelManager);

managerDic.Add(ManagerCode.UI, uiManager);

}

接下来是消息的解析函数：

//接收到消息后解析发送给对应的模块

public void ReveieveMessage(Message message){

BaseManager baseManager=null;

ushort code = message.ID;

//bool IsGet=managerDic.TryGetValue(managerCode, out baseManager);

//if(IsGet==false) Debug.Log("未找到对应的ManagerCode");

////baseManager.ReceieveMessage();

if (code < 5000) baseManager = audioManager;

else if (code >= 5000 && code < 10000) baseManager = cameraManager;

else if (code >= 10000 && code < 15000) baseManager = dataManager;

else if (code >= 15000 && code < 20000) baseManager = modelManager;

else if (code >= 20000 && code < 25000) baseManager = uiManager;

if (baseManager == null){

Debug.LogError("无此Manager");return;

}

baseManager.ReceieveMessage(message);

}

消息中心我们处理完了，下面是我们以UI模块为例开发的消息传递。

我们需要创建BaseUI类让各个UI下的类继承自BaseUI，方便UIManager来管理。也让各个脚本持有对UIManager的引用。

public enum UICode

{

//这里我们以消息的号码区段来区分UI脚本模块 通常一个脚本200个函数已经足够多了 所以每个脚本给200个号段区间

ShowLoginPanel= ManagerCode.UI,

HideLoginPanel,

ShowRegisterPanel = ManagerCode.UI+200,

HideRegisterPanel

}

public class BaseUI : BaseModule {

protected UIManager uiManager;

protected override void Awake() {

base.Awake();

uiManager = messageCenter.UIMgr;

}

public virtual void OnInit() { }

//注册UI脚本给UIManager进行管理

protected virtual void RegisterSelf(ushort[] msgIds,BaseUI baseUi){

for (int i = 0; i < msgIds.Length; i++){

uiManager.AddUIDic(msgIds[i],baseUi);

}

}

protected virtual void SendMessage(Message message ){

uiManager.SendMesssage(message);

}

public virtual void OnReceieveMessage(string data){}

}

我们看到我们在UI基类里定义了一个枚举UICode，然后添加UI脚本里的函数名称为枚举成员。这样我们就可以根据枚举来获取执行的函数。这样有一个缺点，就是UI下的所有脚本函数名不能有重名的。想了好久也没想出其他解决方案。所有事情都有两面性，虽然这里我们需要格外设置枚举成员，但是在消息分发处理的时候确实是非常的方便。完全不需要我们去关心处理。这个类似于中国移动的手机号，我们根据手机号就能分清他是哪个地区的。这样我们在其他类里想要调用UI下注册类里显示面板的函数，我们就发送对应的消息就可以了，非常智能！

在开发各个UI脚本时，通常我们以面板为模块进行开发管理。脚本继承自BaseUI，然后在脚本刷新的时候注册自己到UIManager中。注册的代码也很简单：

public class RegisterPanel :BaseUI {

void Start(){

OnInit();

}

//这里我们要写一个初始化函数 因为并不是所有面板都是在游戏开始被实例化出来的 所以有的脚本start函数并不能执行 需要在实例化时调用并注册

public override void OnInit(){

ushort[] msgIds = new ushort[]{

(ushort)UICode.ShowRegisterPanel,//在这里我们注册我们RegisterPanel面板上的两个函数事件

(ushort)UICode.HideRegisterPanel

};

RegisterSelf(msgIds, this);

}

public override void OnReceieveMessage(string data){ }

public void ShowPanel(){

gameObject.SetActive(true);

}

public void HideRegisterPanel() {

//gameObject.SetActive(false);

Debug.Log("HidePanel");

}

}

下面是我们在登陆面板上调用显示注册面板代码：

//点击注册按钮响应

public void OnRegisterClick() {

Message message=new Message((ushort)UICode.ShowRegisterPanel);

SendMessage(message);

}

因为我们可能在调用其他函数时需要传入参数，所以我们定义自己的数据类型，代码如下:

public class Message

{

private ushort id;

private object[] parameters = null;//将需要传入的参数打包成数组

public ushort ID { get { return id; } }

public object[] Parameters { get { return parameters; } }

public Message(ushort id){

this.id = id;

}

public Message(ushort id,object[] parameters){

this.id = id;

this.parameters = parameters;

}

}

两种消息构造方式，我们可以自由发送有参和无参的消息。我们解析消息中心发送过来的消息是在对应的Manager类里解析的。我们Message.ID强转成对应的UICode，然后根据UICode的号段值获取到注册该UICode的类以及通过反射的方法获取到要调用的函数名。这样一个完整的消息发送系统就已经完成了。代码如下：

public class UIManager : BaseManager

{

private Transform Canvas;

private ushort uiID;

public UIManager(MessageCenter messageCenter) : base(messageCenter) {

Canvas = GameObject.Find("Canvas").transform;

uiID = (ushort)ManagerCode.UI;

}

public Dictionary<ushort, BaseUI> uiDic = new Dictionary<ushort, BaseUI>();

public void AddUIDic(ushort uiCode, BaseUI baseUi) {

if (uiDic.ContainsKey(uiCode)){

Debug.LogWarning("已经注册此uicode,确认是否设置正确！");

return;

}

uiDic.Add(uiCode, baseUi);

}

//处理解析code

public override void ReceieveMessage(Message message) {

BaseUI baseUi;

ushort code = message.ID;

bool IsGet = uiDic.TryGetValue(code, out baseUi);

if (IsGet == false) {

Debug.LogError("没有此code");

return;

}

UICode uiCode = (UICode)code;

string methodName = Enum.GetName(typeof(UICode), uiCode);

MethodInfo mi = baseUi.GetType().GetMethod(methodName);

if (mi == null){

Debug.LogError("此脚本中没有方法！");

return;

}

mi.Invoke(baseUi, message.Parameters);

}

//每个UI面板带的类名和面板名字必须相同

private Dictionary<string, BaseUI> uiPanelDic = new Dictionary<string, BaseUI>();

public BaseUI GetUIPanel(string uiName) {

BaseUI baseUi;

bool IsGet = uiPanelDic.TryGetValue(uiName, out baseUi);

if (IsGet)

return baseUi;

string path = @"Panel/" + uiName;

baseUi = (GameObject.Instantiate(Resources.Load(path)) as GameObject).GetComponent<BaseUI>();

baseUi.OnInit();

baseUi.transform.SetParent(Canvas, false);

uiPanelDic.Add(uiName, baseUi);

return baseUi;

}

public override void SendMesssage(Message message) {

ushort id = message.ID;

//这里需要开发者根据panel进行分段整理 当我们发送的消息时，要检测消息是否属于UI区间段

//当属于UI区间段 我们需要判断该UI区间的模块是否被实例化出来 如果没有 则实例化出来

//当不属于UI区间段 就发送出去 不做处理

RegisterUnRegisterPanel(message);

base.SendMesssage(message);

}

/// <summary>当消息发送给未注册的面板时 在此处要主动注册被调用的面板</summary>

private void RegisterUnRegisterPanel(Message message) {

BaseUI baseUi = null;

ushort id = message.ID;

if (uiDic.ContainsKey(id) == false) {

string uiName = "";

if (id >= uiID && id < uiID + 200) uiName = "LoginPanel";

else if(id >= uiID + 200 && id < uiID + 400) uiName = "RegisterPanel";

else uiName = "";

if (uiName != "") {

baseUi =GetUIPanel(uiName);

if(!uiDic.ContainsKey(id))

uiDic.Add(id, baseUi);

}

}

}

}

这样我们多个开发者就可以并行独立开发，互不影响。但是我现在想到的一个缺点就是代码里函数体会增多，再就是需要设定一个模块下的脚本不可以有同名函数。需要指定被调用函数名为对应的枚举成员。好了，以上就是我个人想到的基于移动通信的号段区分消息发送机制框架。

---------------------

原文：https://blog.csdn.net/qq\_33994566/article/details/79785240