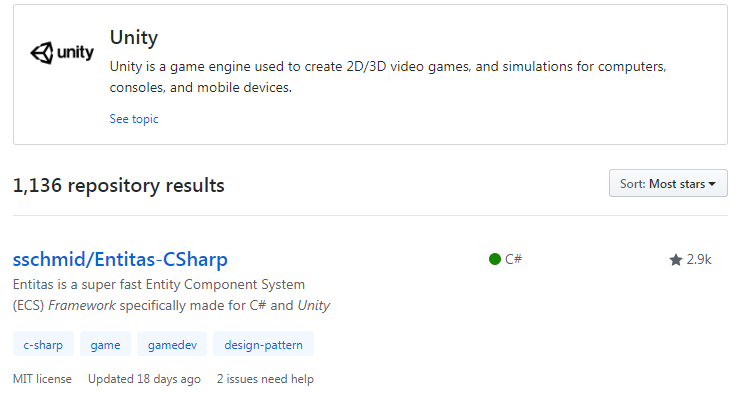
# **游戏架构中的“万金油”**

作者：[ArtStealer](http://gad.qq.com/user/index?id=86976)

链接：<http://gad.qq.com/article/detail/287177>

游戏框架这个话题有点大，游戏开发老司机（团队/个人）基本都有一套自己的框架设计。很多优秀的框架针对自己主要开发的游戏类型都进行的深度的定制和优化。  
一套**游戏框架**也很难适配纷繁复杂的游戏类型。即使更底一层的**游戏引擎**都对游戏有不同的侧重点。  
  
Github上搜索Unity Framework就有1000+的数据，其中最火的是Unity最新提出的ECS框架(此处的**ESC**不同于传统的**Entity+Component**,而是**Entity+ComponentData+System**。本文就不在过多介绍了，感兴趣的同学可以自行去官网了解)

一般来说小型休闲游戏直接就采用引擎示例代码的开发模式可以很快的开发出游戏原型，若一开始就采用复制的框架反而降低了开发效率。但是团队开发的大型游戏如果没有一个严格的规范，那后期的开发维护就会陷入研发陷阱（牵一发而动全身，改一个地方引起了很多意料之外的bug，有的甚至是已经无法满足设计后续的需求）。**一个项目开始时，技术的选型，框架的搭建就是考验开发老司机的时候了，对项目的高度理解制定出合适的框架是一个项目的品质和开发周期的强力保障**。  
说了这么多，大家可能会说：  
>“不都是废话吗？不同的产品采用不同的框架这个外行都知道这句话是万金油的答案，问题是我现在也不知道用什么框架来搭建现在的项目，如何从0到1，如何从游戏开发菜鸟成为老司机呢？”

## 有没有****万金油****的框架呢？

论坛里大神云集，在这里我就抛砖引玉和大家探讨一些基础的框架设计：  
为什么要先探讨一些基础设计呢？**“天下武功出少林”**，所有高深的武学也都是在一些基础身法之上演变组合出来的，就像我们计算机世界里最终都是0/1一样。只有对这些基础模式都驾轻就熟后，才能灵活的应对各种开发场景。  
了解了这些基础的模式，相信大家也可以应对市面上80%以上的产品。  
我抽象了一下游戏开发中我们面对的几个主流的问题和通用的解决方案：  
（总结的不到位的欢迎探讨交流）

## 1 模块管理：

所谓模块管理是对一些游戏资源、数据的组织、功能划分、工具开发等，比较笼统。一般会对有一个**Manager/System**来负责一个模块的封装和管理，便于开发使用。  
因此通常我们会在代码中看到各种各样的管理模块，例如：ResourceManager、UIManager、SchedulerManager、NetManager、EventManager、PoolManager等等。  
要开发这些管理模块，自然就少不了大家熟悉的：**单例模式(Singleton Pattern)**，下面列两个Unity中的单例模块：

// CSharp Singleton

public abstract class Singleton<>: ISingleton where T : Singleton<>;

｛

protected static T mInstance;

static object mLock = new object();

protected Singleton()｛｝

public static T Instance

｛

get

｛

if (mInstance == null)

｛

lock (mLock)

｛

if (mInstance == null)

｛

mInstance = SingletonCreator.CreateSingleton<>();

｝

｝

｝

return mInstance;

｝

｝

｝

// Unity MonoBehaviour Singleton

public abstract class MonoSingleton<> : MonoBehaviour, ISingleton where T : MonoSingleton<>;

｛

protected static T mInstance = null;

public static T Instance

｛

get

｛

if (mInstance == null)

｛

mInstance = MonoSingletonCreator.CreateMonoSingleton<>();

｝

return mInstance;

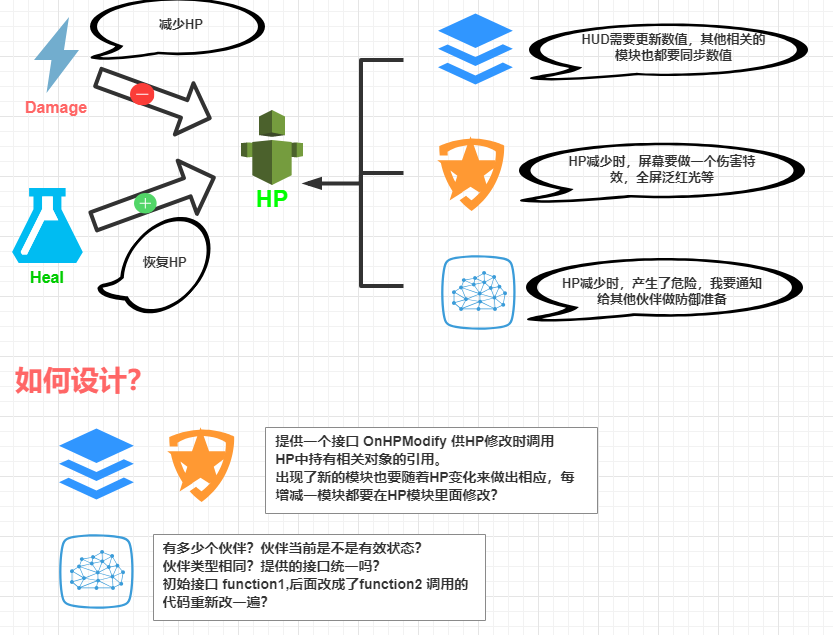
｝

｝

｝

注意此处在多线程实现单例时的**两次判断**。

## 2 模块交互：

交互这块我们学习编程的时候应该就学过程序设计时要 **“高内聚、低耦合”**，围绕这个宗旨软件开发中出现了很多的设计框架，**MVC**是我印象中最早接触的一种框架，虽然说MVC不一定适合于游戏开发，但是其很多扩展和变种模式都多多少少能看到MVC的身影。解决耦合问题我们通常就会用到一个基础的模式：**观察者模式( Observer Pattern)**。  
他是如何来实现解耦合呢？  
  
上面的问题应该很明显了，如果不采用事件系统代码大概是这样

void onHPModify(int value)

｛

hp -= value;

modelA.onHPModify(hp,value);

modelB.onHPChange(value);

EntityManager.onEntityHPChange(this,hp,value);

...

// 全部耦合在一块

｝

如果框架中实现了一个EventSystem,那世界将无限美好，每个对象只要安心的处理自己的事情，并抛出一个Event,关系这个Event的对象自己监听，收到Event后根据自己的需求做对应的处理就好了。

void onHPModify(int value)

｛

hp -= value;

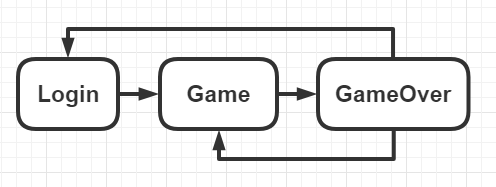
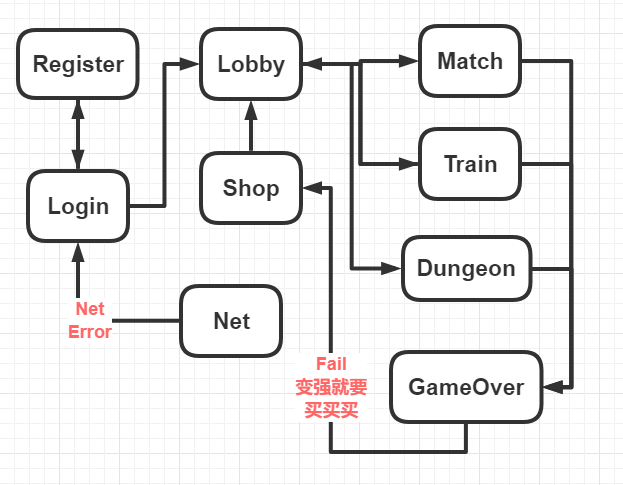
// 干净整洁，都不需要关系外面有多少模块在等着处理这个消息

EventSystem.tiggerEvent(new HPModifyEvent(this,hp,value));

｝

// 其他模块各自监听 HPModifyEvent 进行处理就好了。

## 3 模块转换：

游戏中会根据不同的功能切分成不同的模块，不同模块之间根据不同的条件进行切换。这个地方就要用到**状态模式（State Pattern）**了，游戏中最实用的就是[有限状态机（FSM）](https://mp.weixin.qq.com/s/j4OwzIxb1N1_dy6tZpLrNw" \o "有限状态机（FSM）)、[层次状态机（HFSM）](https://mp.weixin.qq.com/s/Xn9x9Kux9qztpqocFZnAaQ" \o "层次状态机（HFSM）)、[行为树（BehaviorTree）](https://mp.weixin.qq.com/s/q5eXPaGUdRT4G9U2STYmcw" \o "行为树（BehaviorTree）)(行为树框架也很多，腾讯开源的[Behaviac](https://www.behaviac.com/" \o "Behaviac)就是很不错的学习资料)。  
举个简单的例子：  
  
简单的小项目状态不多，你可能用一些简单的**if else, switch** 可以快速的搞定。临时性的增删一个状态，改动成本也不高。  
但是一旦游戏越来越复杂 if else 这种条件判断将会让你对项目的维护力不从心，**每一次流程上的修改都会在内心问候一下策划又瞎改设计，让老子加班 (╯‵□′)╯︵┻━┻**。  
  
上图还算是个非常简洁的图，实际项目的流程状态图要比这复杂几十倍。当然这些大状态的切换其实还不是很复杂因为状态直接关联度不是那么高，实现起来还算简单。如果牵扯到了一些游戏AI那，有可能可能就会用到**层级状态机(HFSM)**甚至**行为树(BT)**了。

## 总结

将框架设计笼统的总结为 **管理、交行、转换** ，并分别介绍了三种对应的设计模式：**单例模式(Singleton Pattern)、观察者模式( Observer Pattern)、状态模式(State Pattern)**。虽然有点片面和简单粗暴，但是为了便于针对性的理解和探讨问题，也不失是一个不错的概括。相信解决了这三个问题基本可以对框架搭建有了一个扎实的基础，上面就可以去根据自身的产品设计去添砖加瓦。**万丈高楼平地起，但是地基的深度决定了楼房的高度，这个地基就是游戏引擎和游戏框架。**  
不知道大家对这个万金油的模式是不是认可呢？欢迎留言或者关注公众号（ArtStealer）探讨。