前言

> 看完SO架构后看了一个开源项目，发现使用的是static数据的方式来处理模块间的通信，然后开始想SO和static的区别，写下了如下的草稿，比较杂乱，内容没有什么营养

SO VS Static

- 访问速度：几乎没有差别
- 工作方式：
  - SO一般需要在Inspector界面指定实例，如果通过脚本实例化后使用无法完成解耦
  - Static变量可直接通过类型找到使用，通常推荐创建一个类的静态实例而不是一个静态类来存储静态变量。也就是回到了单例模式，只要将静态实例置空就可以让其被GC回收
  - Static数据只能存在一份，而SO可以创建多份实例
  - 从风格上讲，全局静态变量更像硬编码，因为全局静态变量的生命周期不由自己控制，不能保证访问的是不是null，而SO是以资产形式引入，可以保证非null
  - SO的代码中依赖的数据比较明显
  - 其实本质上是差不多的，因为对于需要存储的数据，无论是SO还是static都需要一个初始化管理的类，以及保存系统，纯代码可能更直观，SO胜在方便设计

- 使用static的一种实践方式就是让static变量指向引用类型，不需要时置为null，所以需要一个 普通类来存储静态的数据类实例
  - 而这个普通类只需要一个就可以，所以使用 单例模式 （非mono类单例），非MB单例模式的脚本需要通过new产生类实例，然后指定为Instance，为了避免new出多个类实例，一种做法是私有构造函数，然后通过属性在get时调用构造函数创建类实例
    - MB的单例模式：因为MB没有构造函数，一般在Awake阶段指定Instance为当前脚本 或者 在获取类的静态实例时通过AddComponent方式添加一个实例化的MB脚本并指定为Instance
    - 对于其中的static数据实例，我们希望可以 控制其分配堆内存的时机 ，所以不在声明中赋值
    - 对于使用静态数据的其他类，最好不需要检查其是否为null，所以需要一个脚本 提前对静态变量赋值 ，即设计一个初始化脚本显式初始化我们需要的单例类和其中的静态实例
    - 一般的 单例模式就是让这个类成为数据的载体 ，其实例化后就是静态数据实例，需要通过xxxClass.Instance.xxField访问，这里的Instance可以是属性也可以是公开的static字段
    - 将数据类实例静态化其实就是Singleton的核心做法，只不过一般的Singleton会实现很多控制内容，而现在的做法是 数据与逻辑分离 ，公开数据，由各个组件进行逻辑控制
- 另外一种设计是 不设置成为单例类 ，我们其实不需要这个类的实例，只需要提供静态方法控制静态变量的实例化和置空，更方便管理
  - 同样可以设置构造函数为私有 防止实例化
- 使用static的另一个不方便的问题在于 测试 ，SO可以可视化看到数据，而且可以在编辑器面板控制数据，但static不行，需要另外的脚本辅助
  - 传统的不使用Static的说法还有另一个原因： 多线程不安全 。但是Unity是单线程的，（如果不使用多线程）不需要考虑这个