StrangeIoc的理解

1. 什么是StrangeIoc

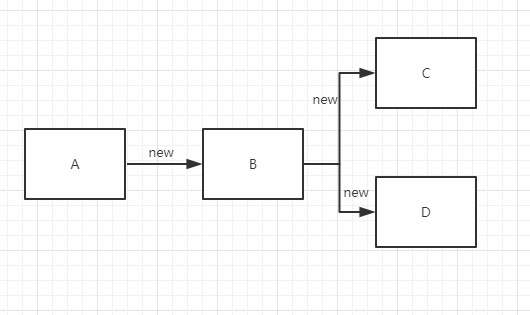
说的比较多的一句话是，一个针对Unity3，超级轻量、高度可扩展的、IOC的框架。

已知公司内部使用StrangeIOC框架的游戏有：腾讯桌球、欢乐麻将、植物大战僵尸Online，数据来自KM。

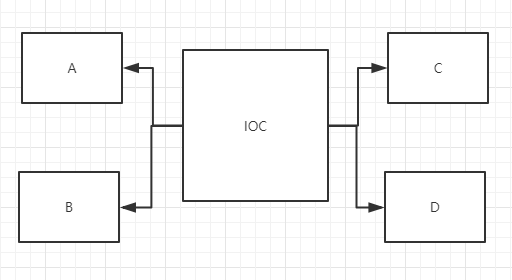
1. 什么是IOC

Ioc：Inversion of Control （控制反转）

为什么有这个?先看下原来的对象管理方式，面向对象开发过程会有很多对象。当A需要B时，会new一个B，当B需要C和D时同样会new出来，这样所有对象都是耦合比较重的。



通过IOC创建的对象



所有实体的管理都在IOC这个容器中完成。那么A是怎么获得B这个对象的呢？

这里有另一个概念DI(Dependency Injection依赖注入)。

通过注入的方法把对象设置进来。

我们看下代码中我们要怎么做：

Class A {

[Inject]

public B b {get;set;}

}

Class B {

[Inject]

public C c {get;set;}

[Inject]

public D d {get;set;}

}

这句话表示我需要一个B的对象，[Inject]标签表示需要通过注入的方式来赋值，实际上在这个脚本创建出A对象的同时，B对象就会在IOC容器中去找是否已经创建成功，如果创建成功就会赋值到变量上，如果没创建就会先创建再赋值。

看下IOC做了什么呢？

实际上是个容器，所有的实例对象都在IOC这个容器中产生。

1. 框架介绍

StrangeIoc采用MVCS结构，Model、View、Controller、Services。

有一个MVCSContext脚本实际对应的就是IOC对象。

View:包含View和Mediator。

View主要处理界面相关逻辑，提供修改界面接口给Mediator。

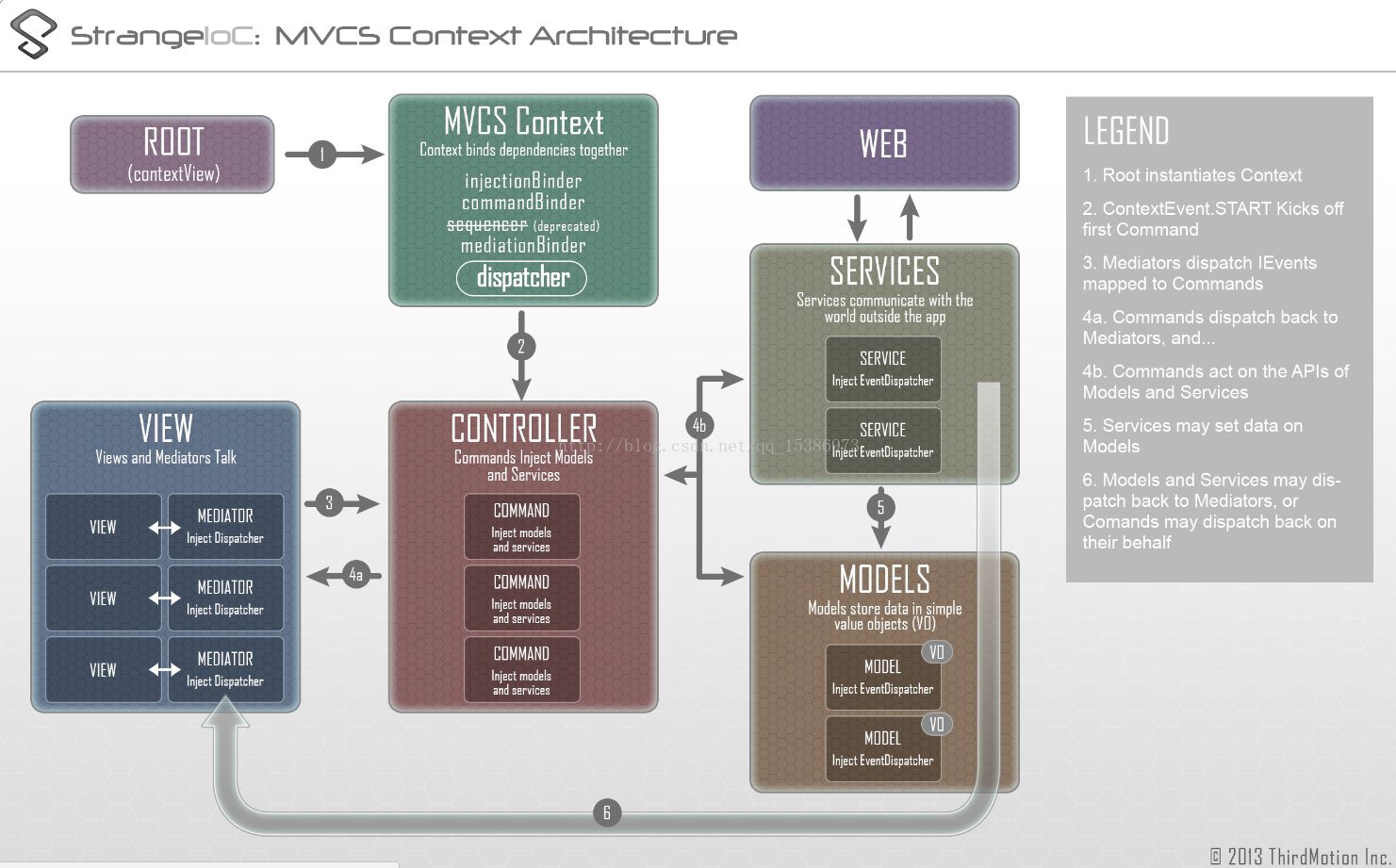
Mediator与Controller之间通过事件交互，并调用View的界面修改接口。

Model:数据层

Services:网络层

Controller:持有Model、Services。与Mediator通过事件交互。

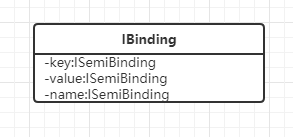
附一张官方介绍图：



1. 核心类介绍

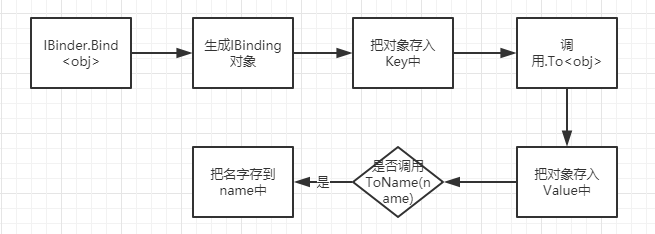
IBinder 对外的接口

IBinding 内部绑定实体类 有3个封装的数组对象组成，分别为Key，Value，Name数组



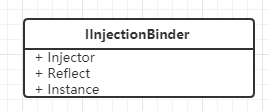
一个完整的绑定流程:

injectionBinder.Bind<IExampleModel>().To<ExampleModel>();



DI相关的接口

IInjectionBinder



IInjector

如果对象没有初始化在这里也进行实例生成

获取Reflect结果复赋值到对应的Inject标签的值上

IReflectionBinder

1.找到[Constructor]标签的构造方法

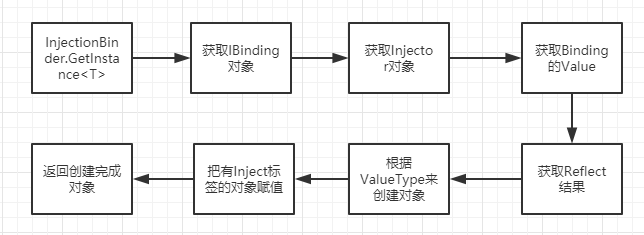
2.找到[Inject]标签的属性

    1.找到私有Inject标签会报错提醒

2.保存对应名字，类型，属性到数组中

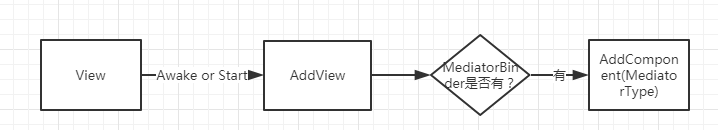
一个完整的初始化流程

injectionBinder.GetInstance<ICommandBinder>();



View与Mediator初始化流程

mediationBinder.Bind<MainView>().To<MainMediator>();



1. 用例

一艘飞船，需要用到武器，这个武器到底是什么武器呢？

class Spaceship : ISpaceship

{

public void input(float angle, float velocity)

{

//do stuff here

}

[Inject] //<----- The magic word!

public IWeapon weapon{get;set;}

}

injectionBinder.Bind<ISpaceship>().To<Spaceship>();

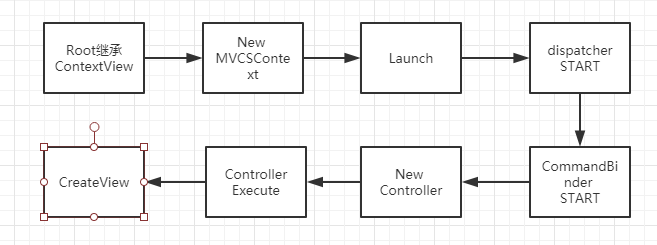
injectionBinder.Bind<IWeapon>().To<PhaserGun>();（激光枪）

injectionBinder.Bind<IWeapon>().To<SquirtCannon>();（散弹炮）

这样就能很方便的只需要Value对应不一样的值就可以很快的初始化不同的武器了。

当然重复绑定是不允许的，先要UnBind，然后重新Bind就可以了。

1. 框架启动



会抛出固定的Start事件。支持事件方式启动框架。

1. 框架运用

出发点：

1. 框架对于项目的应用
2. 对象的生命周期管理
3. 性能问题

针对问题解决方案：

1.我们现在项目中系统框架：

Form：界面相关操作

Module:网络数据处理、本地数据处理、游戏逻辑处理等复杂逻辑。而且这个Module还是1对多个Form，通过partial来让这个module的复杂度增加到很难阅读的程度。

这种设计的缺点：

1. 可以处理界面逻辑的地方不清晰，有些模块在View中，有些在Module中。
2. 一个大的系统包含的子系统非常多，通过partial来写可能导致一个Module有上万行。当需要找文件的时候异常辛苦，因为这种情况下文件名没任何意义，一般建议文件名跟类名一致会方便阅读。
3. Module过于重度，导致后期重构是一个很大的难题。

最快的框架引入及重构方案：

通过MVC来重构我们现有的代码：

View:写界面相关逻辑

Controller:写与界面无关的逻辑，与model交互。

Model：网络交互，数据持有化。

好处：

1. View与Controller为1对1的关系，这样能很方便的找到View对应的功能逻辑，需要改时也只是系统内的改动。
2. Model功能单一，看上去会更清晰。协议错误，数据没有清理或赋值只需要去model中看即可。
3. 方便后续修改扩展。

从现有框架调整为这种方式成本很低，可以让后续的模块按这种方式来做。引入的只是一个MVC思想，并没有外部组件，没有其他风险。比较推荐。

通过StrangeIOC来重构我们项目：

View：处理界面操作逻辑，提供界面操作接口。

Mediator：与Controller通过事件来交互，由于持有View，界面交互操作在这里绑定，如按钮功能。调用View的界面操作接口。

Services：处理网络数据，协议收发。

Model：处理本地数据，配置读取

Controller：整合本地网络数据，与Mediator通过事件交互

好处：

1. 模块划分更加清晰，很容易找到问题所在。
2. 模块间没有任何耦合关系，用到的对象都可以从IOC中获取。

3）方便大项目后期的维护扩展。

坏处

1. 有学习理解成本
2. 用到了反射，对性能有影响
3. 简单问题复杂化，一个很简单界面也需要用到MVCS

感觉项目后期引入这个框架不太适合，因为IOC的引入要把引用改到IOC中成本会比较大，还是建议在项目启动初期能引入这个框架会比较合适。

接下来会了解ECS框架，到时候根据ECS的特点再进行一次对比，看Unity官方展示视频同频3W个单位还能保证30帧，到5W多单位时会降到15帧左右。感觉很强大。

针对现有IOC框架进行优化：

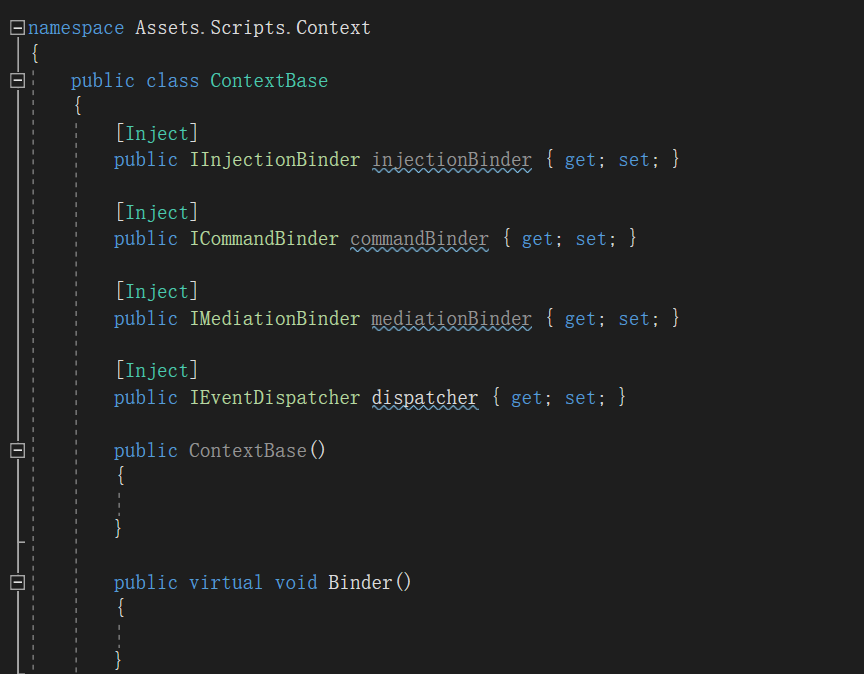
从官方给的框架图可以看出来，建议我们把对象都放在IOC中，所以势必会造成这个MVCSContext非常大，因为一个游戏会有非常多的模块，如果全写在一个文件内就会造成这个文件非常庞大。所以我建议增加2个方法来分担MVCSContext的负担。

在MVCSContext增加2个方法：

addModuleComponents() 初始化模块组件

InstantiateModuleComponents() 实例化模块组件

定义一个模块Context父类，它含有MVCSContext的绑定功能，具体代码如下：



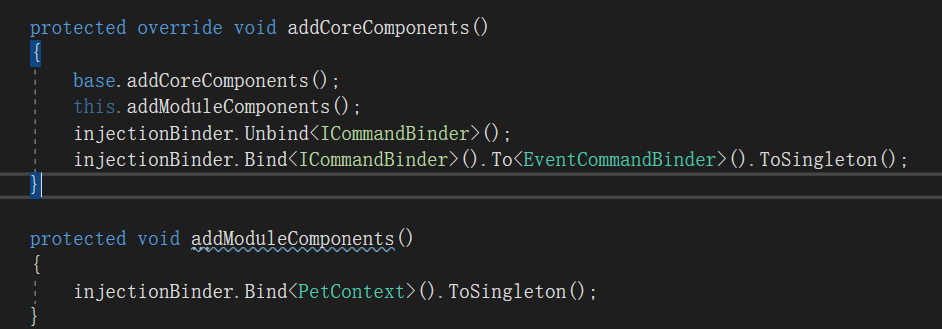
比如我们有一个宠物系统，比较大，包含很多界面功能，我们可以这样做：

1>声明一个PetContext用于绑定这个系统用到的一些对象信息

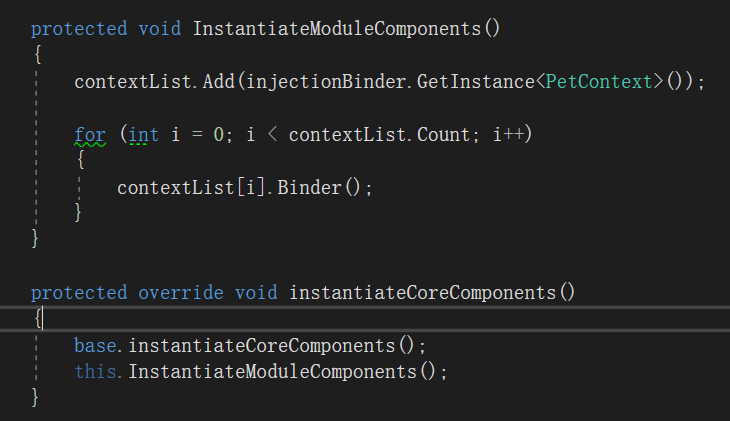
2>初始化并实例化对象。

看下具体代码：

初始化，在核心组件初始化完之后：

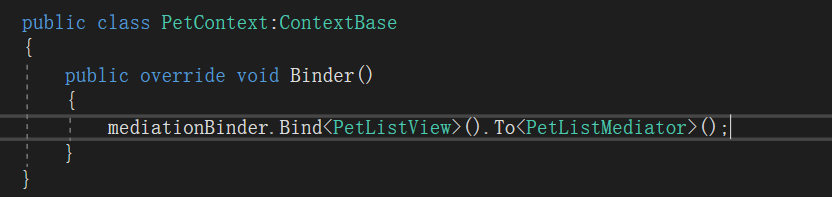


实例化，在核心组件实例化完之后：



通过GetInstance来创建对象会默认把Inject标签对象赋值。

具体绑定逻辑由对应模块的Context也就是IOC对象来完成。



这样做以后模块划分会更清晰，虽然对象还是持有在总的IOC对象对象中，但是阅读、修改的成本会降低很多。

2.对象生命周期

**UI模块**

只存在绑定关系，当出现绑定有mediationBinder时，会创建View自动加上mediator。IOC负责创建对象但并不持有，通过引用计数没有引用时自动GC。

**对象模块**

对象的创建分为多种情况，对象分为3种：

1. 普通对象

由IOC负责创建，但是不负责销毁，销毁由引用计数消失GC自动回收

1. 单例对象

由IOC对象创建，创建完成后对应的value会有Type变为Object，此后访问value时不用创建直接返回即可。一直被持有在injectionBinder的value值中，需要UnBind把引用释放，通过GC来回收。

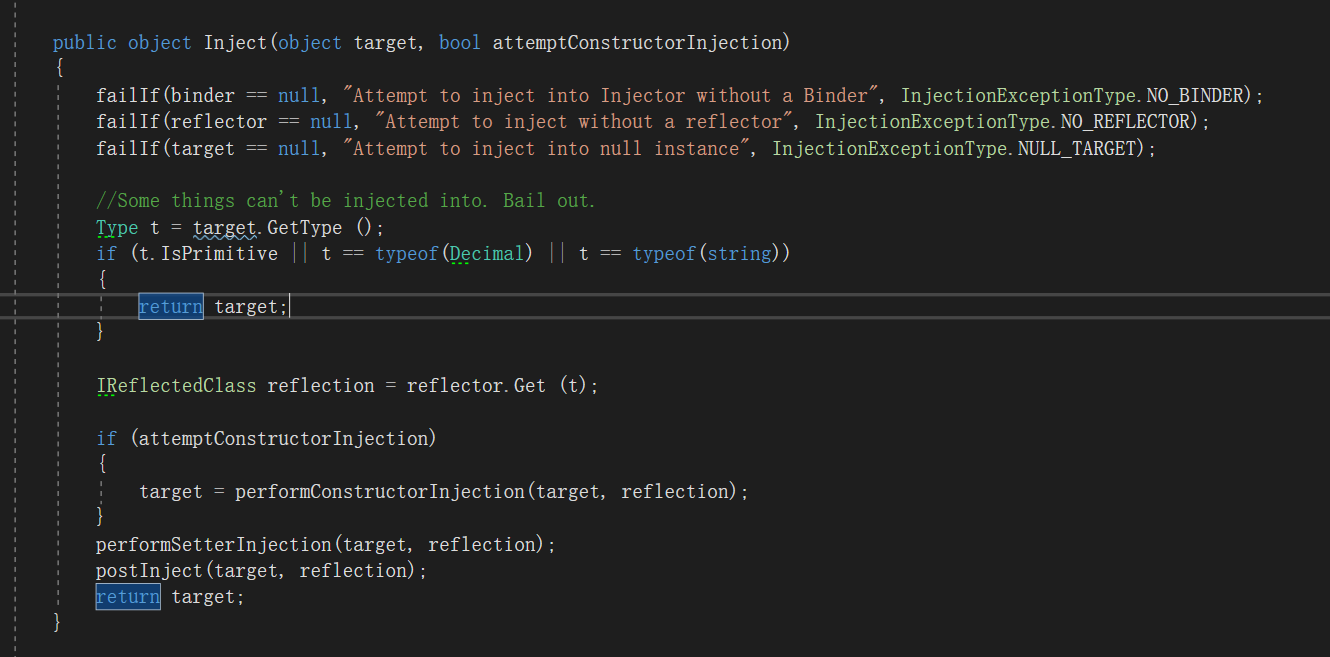
1. 值对象

通过接口ToValue来绑定Value对应的值，通常这个值在外部已经创建好了，只是放在IOC中来管理。比如GetComponent一个对象直接存为值对象。

当需要对象时，直接通过IOC的GetInstance来创建，或者通过Inject由IOC来给你赋值，赋值的来源可能是上面的3种途径。这样我们在与多模块交互时，只需要把这些变量放在IOC容器中来管理即可。

4.性能分析

我们看下Inject的代码



这里是Inject的核心：

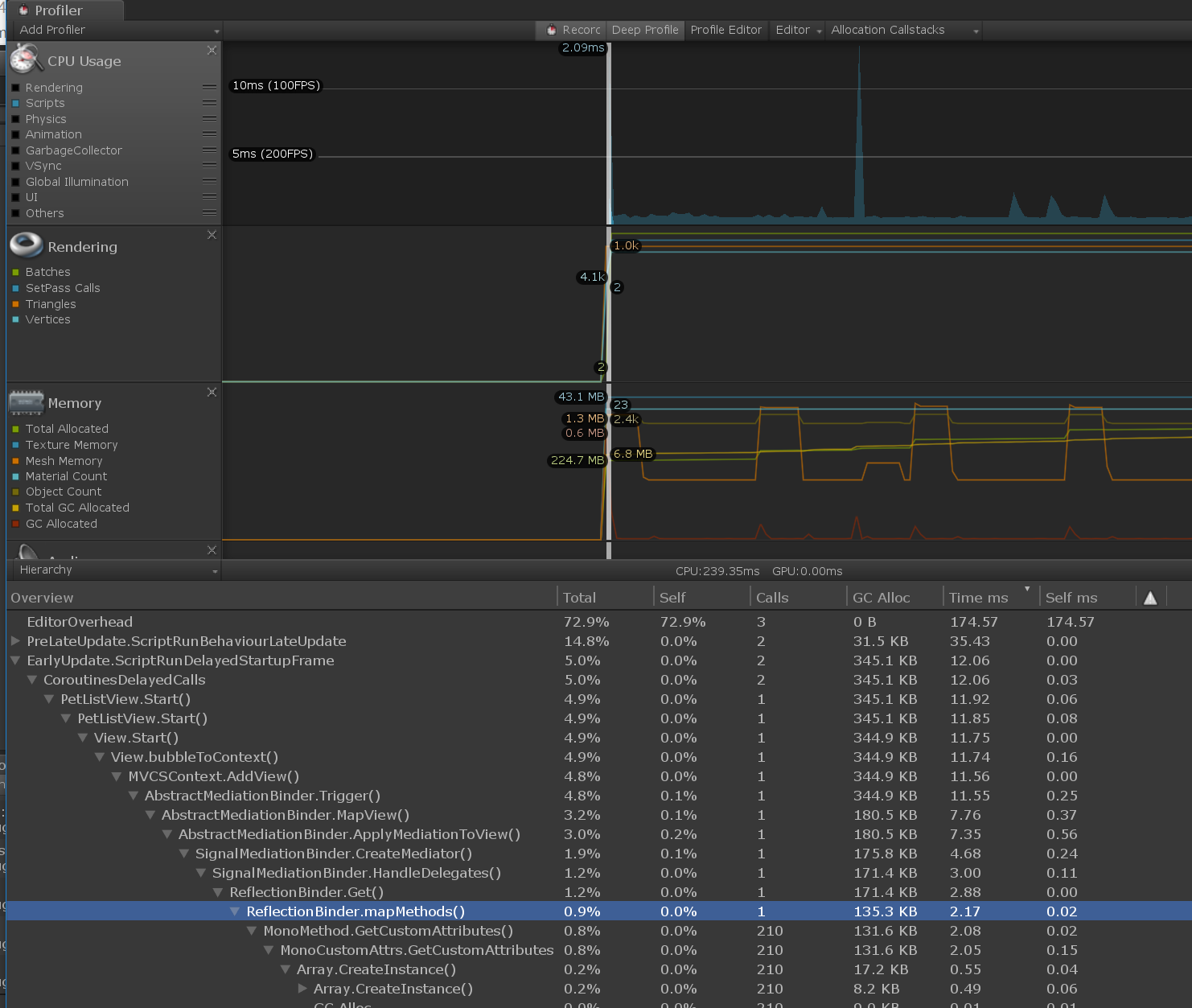
Reflector功能前面已经说过了，主要是找对象的标签特性，并且缓存起来。

performConstructorInjection通过 构造函数来注入，如果没有加这个标签会默认调用无参构造函数，标签格式[Construct]，这种方式注入好处就是

performSetterInjection注入有Inject标签属性的变量 如果这里用的对象不存在也会被实例化出来。

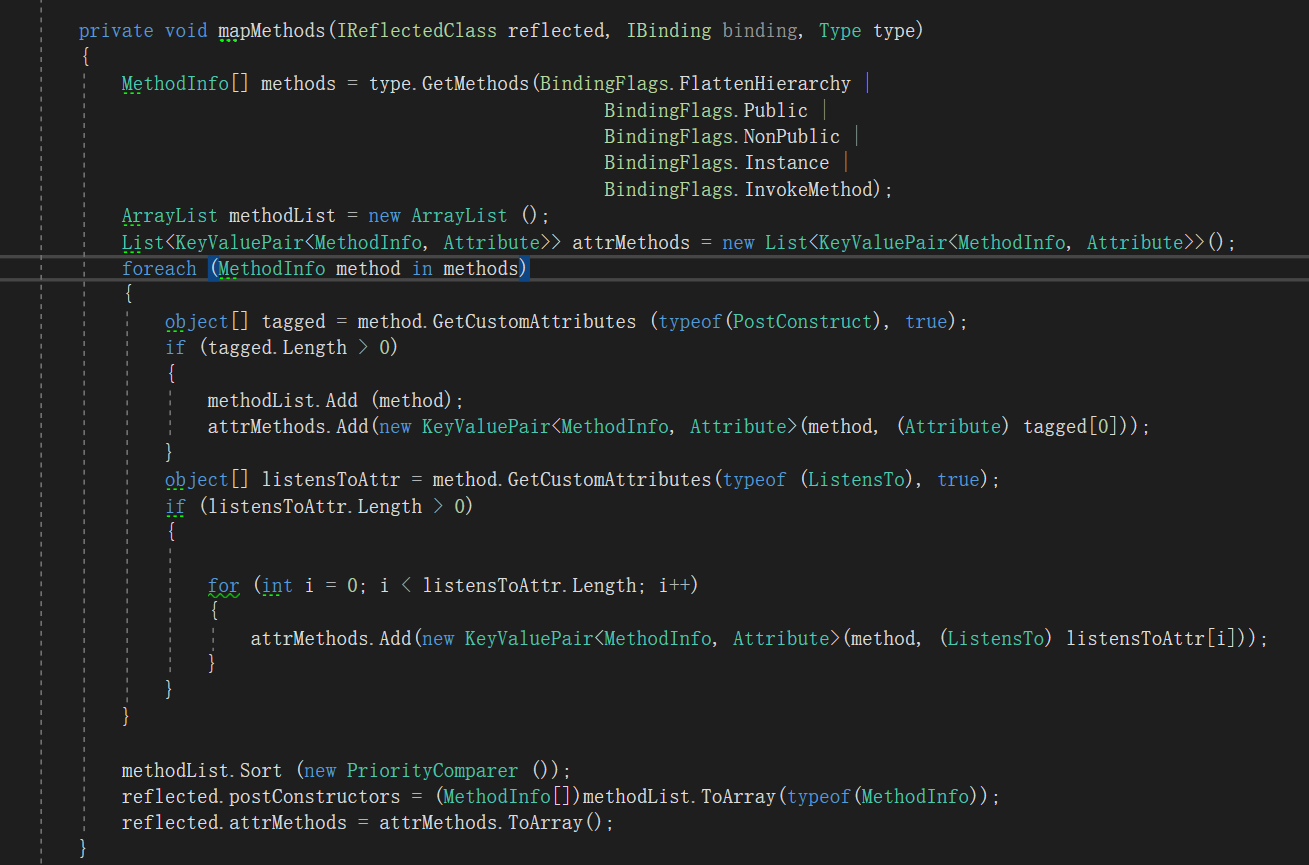
postInject

启动框架并初始化一个系统：



深度分析的情况下反射导致GC的产生会比较多，但是耗时并不高。

对应mapMethods方法：



1. 总结

相比于MVC这个框架的结构划分更清晰。我们只用关注逻辑不用关注对象产生。

项目启动初期可以选StarngeIOC框架，以IOC、DI的思想来设计，转变之前的一些开发方式。后期扩展、维护的工作都不大，还是比较推荐的。

对于一些简单的系统可以只提供View和mediator，因为mediator和controller通过事件交互，可以很灵活的由外部模块的controller进行调用。

大系统还是建议controller调用自己的mediator，需求操作其他模块是也是controller先调用其他模块的controller。