IOC的概念：

IOC是Inversion of Control的缩写，多数书籍翻译成“控制反转”。

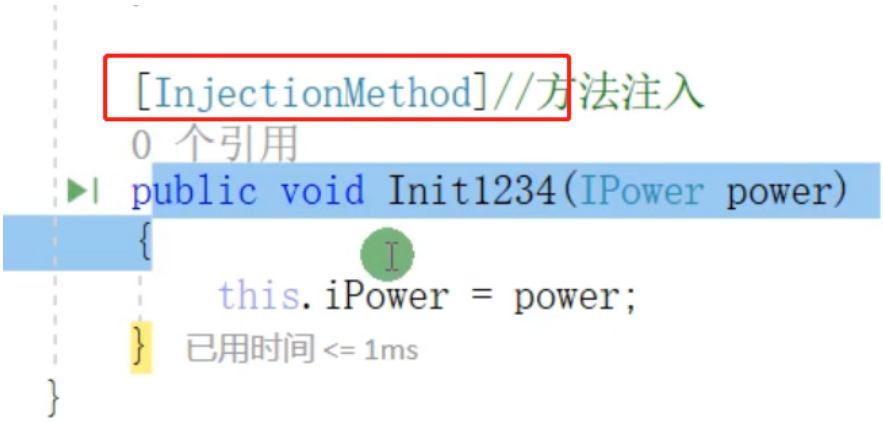
1996年，Michael Mattson在一篇有关探讨面向对象框架的文章中，首先提出了IOC 这个概念。对于面向对象设计及编程的基本思想，前面我们已经讲了很多了，不再赘述，简单来说就是把复杂系统分解成相互合作的对象，这些对象类通过封装以后，内部实现对外部是透明的，从而降低了解决问题的复杂度，而且可以灵活地被重用和扩展

一：Unity的依赖注入

构造函数注入（构造函数注入是最好用的，因为可以不依赖框架上面的[Injectionconstructor]会自动的找到参数最多的构造函数 ）


属性注入：


方法注入




- [ ] 如果是用Unity的配置管理来进行依赖注入的话，可以不引用对应的dll，只好保证对应的dll在通过一个目录下面即可，因为Unity调用的方式的本质本来的就是反射

二：声明周期管理

下面的这种方法实现容器单例

```
rType<IPhone, AndroidPhone>(new ContainerControlledLifetimeManager());//容器单例  单例就不要自己实现
```

下面这种是线程单例：即是同一个线程里面就是单例的，不是同一个线程里面就不是单例的。用回调线程是调用线程是同一个线程来证明。

```
container.RegisterType<IPhone, AndroidPhone>(new PerThreadLifetimeManager());//线程单例
```

分级容器：同一个容器创建的就是单例，不同容器创建的就是其他单例

```
container.RegisterType<IPhone, AndroidPhone>(new HierarchicalLifetimeManager());//分级容器单例
IUnityContainer childContainer = container.CreateChildContainer();//获取子容器
```

外部可释放单例：外部释放了，但是内部还是单例

```
container.RegisterType<IPhone, AndroidPhone>(new ExternallyControlledLifetimeManager());
```

循环引用创建单例

```
ainer.RegisterType<IPhone, AndroidPhone>(new PerResolveLifetimeManager());//循环引用 不推荐
```