DI（DependencyInjection）和IoC（Inversion of Control）的区别

- 控制反转：控制权由应用代码中转到了外部容器
  - 我们直接在对象内部通过new进行创建对象，是程序主动去创建依赖对象；

    而IoC是有专门一个容器来创建这些对象，即由Ioc容器来控制对象的创建；（不一定，可以外部创建实例然后注册到IoC容器中，但是查找依赖对象肯定是通过IoC容器）

    谁控制谁？当然是IoC 容器控制了对象；

    控制什么？那就是主要控制了外部资源获取（不只是对象包括比如文件等）

    2分钟带你理解IOC - 知乎 (zhihu.com)

    用到的核心技术就是：反射
- 依赖注入：组件之间的依赖关系由容器在运行期决定，即由容器动态地将某种依赖关系注入到组件之中。
- IoC和DI有什么关系呢？

  IoC和DI其实本质相同，但是IoC实现通过IoC容器

  IoC容器是实现依赖注入的方式之一，而SO也是实现依赖注入的一种方式

  【CodeTrick】1.对IOC与DI的理解_PartnerLv的博客-CSDN博客

依赖注入的实现

- 常用的依赖注入框架
  - .net core自带的DI
  - Unity（UnityContainer）（已经不再维护了）
  - zenject（beat saber/pokemon go）（轻量级的高性能依赖注入框架）
  - StrangeIoc
- 注入方式

  构造函数注入

  字段注入

  属性注入

  方法注入

依赖注入的必要性

- Unity 3D开发有没有必要使用依赖注入？ - 知乎 (zhihu.com)
- 依赖注入控制反转在游戏开发中有实际意义吗？ - 知乎 (zhihu.com)
  - 有个回答提到了：
    - 某些数据的最终处理者，不会发生变化，那么就不必用，比如日志文件的生成，就是固定写入某个文件的。 直接static模式都可（使用单例就可以）
    - 这也可以用来判断使用SO和使用Singleton（static数据）的合适时机
    - 这也点出了Static（单例）硬的特征，和SO软的特征，Static适用于明确的数据/方法，SO适用于可能变化的数据/方法
      - SO的软更多体现在编辑器内，可以赋予一个对象同一个类型SO的不同实例（以unite 2017演讲中的`石头剪刀布`为例），当然运行时代码改变SO实例也是软的一部分，可一般不会对SO这么做
      - 而传统的依赖注入框架（IoC等）更多是在代码层面的软，一个数据处理者是可以通过IoC容器动态获取的（可以改变获取到的内容）
      - 因为SO更多是在编辑器内的赋值，所以看起来比较像static，但对于动态赋值的SO，不能用static替代
- The truth behind Inversion of Control - Part I - Dependency Injection - Seba's Lab (sebaslab.com)

    - 文章中提到了注入太多依赖也不好
    - 注入太多依赖的问题大多数来自于`破坏了单一职责原则`，否则一个类应该只有少数的几个依赖，此时需要做的是模块化
    - IoC不适合开发游戏
    - 通信的方式
        - 直接引用，A依赖B，A中直接调用B的方法（A中注入B）【不好】
        - 直接的事件监听，A依赖B的事件，A监听B的事件（A中注入B）【不好】
        - 命令模式：A和B解耦，A能通过命令调用B的方法
        - 中介模式：A和B解耦，A和B都被另外的一个中介者管理（例如UI的顶级组件管理子级UI组件）
        - 其他（观察者/事件总线/事件队列）：A和B都不知道对方，处理方式类似中介模式
    - 一个观察到的现象，动态创建的物体一般不会成为被依赖的对象，所以工厂可以持有动态对象的依赖
    - 依赖倒置原则很重要
        - 高层不依赖低层，两者都依赖抽象（OOP中就是接口，ECS中就是Component）
        - 抽象不依赖细节
        - 如果处处使用可以依赖注入获取依赖，而不分层，代码同样会变得很乱。
        - 举例来说，health是更高层的抽象，应该让health寻找entity（IHaveHealth），而不让entity拥有health(好莱坞原则)
        - entity（低层）通过接口向高层传递自身，不必去寻找高层抽象的依赖
- 注入太多依赖的下场（SO同理），造成如下原因是1、没有实现单一职责 2、没有Inversion of Flow Control的概念
    -
```
public sealed class DoingSomethingBad
{
    [Inject] public ISimulationFactory          simFactory              { private get; set; }
    [Inject] public ICatalog                    typeInventory           { private get; set; }
    [Inject] public IMonoBehaviourFactory       monoBehaviourFactory    { private get; set; }
    [Inject] public IMachineMap                 playerMap               { private get; set; }
    [Inject] public PlayerBuiltListener         playerBuilt             { private get; set; }
    [Inject] public WeaponFireManagerFactory    weaponFireManagerFactory{ private get; set; }
    [Inject] public ratingData                  rating                  { private get; set; }
    [Inject] public HealthStatus                health                  { private get; set; }
    [Inject] public HealthStatusContainer       healthStatusContainer   { private get; set; }
    [Inject] public PlayerMachinesContainer     playerMachinesContainer { private get; set; }
    [Inject] public LobbyGameStartPresenter     lobbyGameStart          { private get; set; }
    [Inject] public GroundHeight                groundHeight            { private get; set; }
    [Inject] public GameObjectPool              pool                    { private get; set; }

    public void FunctionWithResponsability1()
    {
        // ...
    }

    public void FunctionWithResponsability2()
    {
        // ...
    }

    public void FunctionWithResponsability3()
    {
        // ...
    }

    public void FunctionWithResponsability4()
    {
        // ...
    }

    public void FunctionWithResponsability5()
    {
        // ...
    }
```

```
    public void FunctionWithResponsability6()
    {
        // ...
    }
}
```

Inversion of Flow Control

- 通过事件机制实现
- 通过策略模式
- 通过ecs
- 依赖注入后是让低层的对象处理控制流程（低层依赖高层破坏了好莱坞原则，inspector面板中拖拽So实例就是表现），而控制流程反转就可以避免依赖注入
- 具体来说，一个系统（system）遍历所有注册的实体(entities，接口)调用他们的方法（不同的策略）
    - 比如SLG中，场景中敌人的攻击箭头就是通过一个系统调用接口，处理不同AI的搜索下一个攻击对象的方法
- ECS的设计不太确定是否能很好满足各方面的设计需求，有待进一步的探索，但用反转控制流程替代依赖注入很有必要（避免滥用SO）
- The truth behind Inversion of Control – Part V – Entity Component System design to achieve true Inversion of Flow Control - Seba's Lab (sebaslab.com)
    - 文章中链接说所有逻辑由System处理，暂时还未实践ECS，不好下定论