# IOC: Inversion of Control

**Inversion of control**, or **IoC**, is an abstract principle describing an aspect of some [software architecture](http://en.wikipedia.org/wiki/Software_architecture) designs in which the [flow of control](http://en.wikipedia.org/wiki/Control_flow) of a system is inverted in comparison to procedural programming.

Using Inversion of Control this central control as a design principle is left behind. Although the [caller](http://en.wikipedia.org/wiki/Caller) will eventually get its answer, how and when is out of control of the caller. It is the callee who decides to answer how and when.

Inversion of Control as a design guideline serves the following purposes:

* There is a decoupling of the execution of a certain task from implementation.
* Every system can focus on what it is designed for.
* Every system does not make assumptions about what other systems do or should do.
* Replacing systems will have no side effect on other systems.

#### **Implementation techniques**

1. using a factory pattern,
2. using a service locator,
3. using a constructor injection,
4. using a setter injection,
5. using an interface injection, and
6. contextualized lookup.

Compare to layered architecture.

我觉得用[**IOC**](http://www.jdon.com/jivejdon/tags/436)对设计一个具有很大的变数的系统很有用处，缺点是增加了系统的复杂性。

Acegi Security是一个典型的[**IOC**](http://www.jdon.com/jivejdon/tags/436)程序，花了一个星期才弄清楚里面的运作原理，了解了里面各个部分插入之间的关系，而我要求的功能如果全部自己用传统的方法编程实现的话可能速度只要一两天，这是由于Acegi [**IOC**](http://www.jdon.com/jivejdon/tags/436)粒度太细，极大的增加了系统的复杂程度和熟悉成本。

但另外一方面发现以前认为很复杂的几个需求用Acegi只需要插入几个简单实现的class即可实现，对于security这样的需求，每个系统根据业务的不同可能有不少的差别，用 [**IOC**](http://www.jdon.com/jivejdon/tags/436)来实现的话才能满足通用的需求。所以对于code library，IoC是个重要的指标，缺少[**IOC**](http://www.jdon.com/jivejdon/tags/436)就会失去很大的灵活性而失去很多用户。

Ioc和dependency injection关系密切，切断了直接的依赖。

模板方法模式充分的体现了“好莱坞”原则。IOC是Inversion of Control的简称，IOC的原理就是基于好莱坞原则，所有的组件都是被动的（Passive），所有的组件初始化和调用都由容器负责。

所有的framework都是遵循好莱坞原则设计的，否则就不叫framework。framework使用IoC的目的:

1. 对基于接口编程的支持
2. 减少单件和抽象工厂的依赖
3. 降低业务和框架的耦合
4. 业务组件可复用,可插拔

Layered architecture: 上层调用底层；

IOC: Framework调用上层