**简介**

 ABP为创建模块及组织它们提供基础框架。一个模块可依赖于另一个模块。通常地，一个程序集做为一个模块。

**模块定义**

public class MyBlogApplicationModule : **AbpModule**

{

public override void Initialize()

{

IocManager.**RegisterAssemblyByConvention**(Assembly.GetExecutingAssembly());

}

}

模块定义类负责把自己的类注册到依赖注入里（约定做法如上所示）、配置应用和其它模块、为应用添加新的特性等...

**生命周期方法**

ABP按依赖关系顺序调用这些方法。如果模块A依赖于模块B，模块B会在模块A前初始化，确切的启动方法顺序是：预初始化B，预初始化A，初始化B，提交B和提交A。

**PreInitialize（预初始化）**

应用启动时最先调用这个方法，它通常在初始化前configure（配置）框架或是其它模块。

**Initialize（初始化）**

依赖注入注册一般都在这里完成，一般都使用IcoManager.RegisterAssemblyByConvention方法。

**PostInitialize（提交初始化）**

在启动过程中最后调用这个方法，此时可以安全的解析依赖。

**Shutdown（关闭）**

当应用关闭时调用这个方法。

**模块依赖**

一个模块可以依赖于其它模块，它要求使用DenpendsOn特性显式地声明依赖关系，如下：

**[DependsOn(typeof(MyBlogCoreModule))]**

public class MyBlogApplicationModule : AbpModule

{

public override void Initialize()

{

IocManager.RegisterAssemblyByConvention(Assembly.GetExecutingAssembly());

}

}

**插件模块**

尽管ABP解析是从startup模块开始到所有依赖，但也可以动态加载模块。AbpBootstrapper类定义PlugInSpurces属性，该属性用来就是用来添加源到动态加载插件模块，一个插件源可以是任何实现了IPlugInSource接口的类，PlugInFolderSource类实现了这个接口，并指明在某个文件夹下的程序集里获取一个插件模块。

在经典的Asp.net Mvc应用里，我们可以重写global.asax里的Application\_Start来添加文件夹插件，如下所示：

public class MvcApplication : AbpWebApplication<MyStartupModule>

{

protected **override void Application\_Start**(object sender, EventArgs e)

{

**AbpBootstrapper.PlugInSources.AddFolder(@"C:\MyPlugIns");**

base.Application\_Start(sender, e);

}

}

**自定义模块方法**

你的模块同样可以自定义方法，其它依赖于该模块的模块，就可以使用这些方法。

假设MyModule2依赖于MyModule1，想在预初始化时调用MyModule1里的一个方法。

public class MyModule1 : AbpModule

{

public override void Initialize()

{

IocManager.RegisterAssemblyByConvention(Assembly.GetExecutingAssembly());

}

**public void MyModuleMethod1()**

{

//this is a custom method of this module

}

}

**[DependsOn(typeof(MyModule1))]**

public class MyModule2 : AbpModule

{

private readonly MyModule1 \_myModule1;

public MyModule2(**MyModule1 myModule1**)

{

\_myModule1 = myModule1;

}

public override void PreInitialize()

{

**\_myModule1.MyModuleMethod1(); //Call MyModule1's method** }

public override void Initialize()

{

IocManager.RegisterAssemblyByConvention(Assembly.GetExecutingAssembly());

}

}

此处，用构造器注入MyModule1给MyModule2，所以MyModule2可以调用MyModule1的自定义方法，这仅在Module2依赖于Module1的情况下可用。