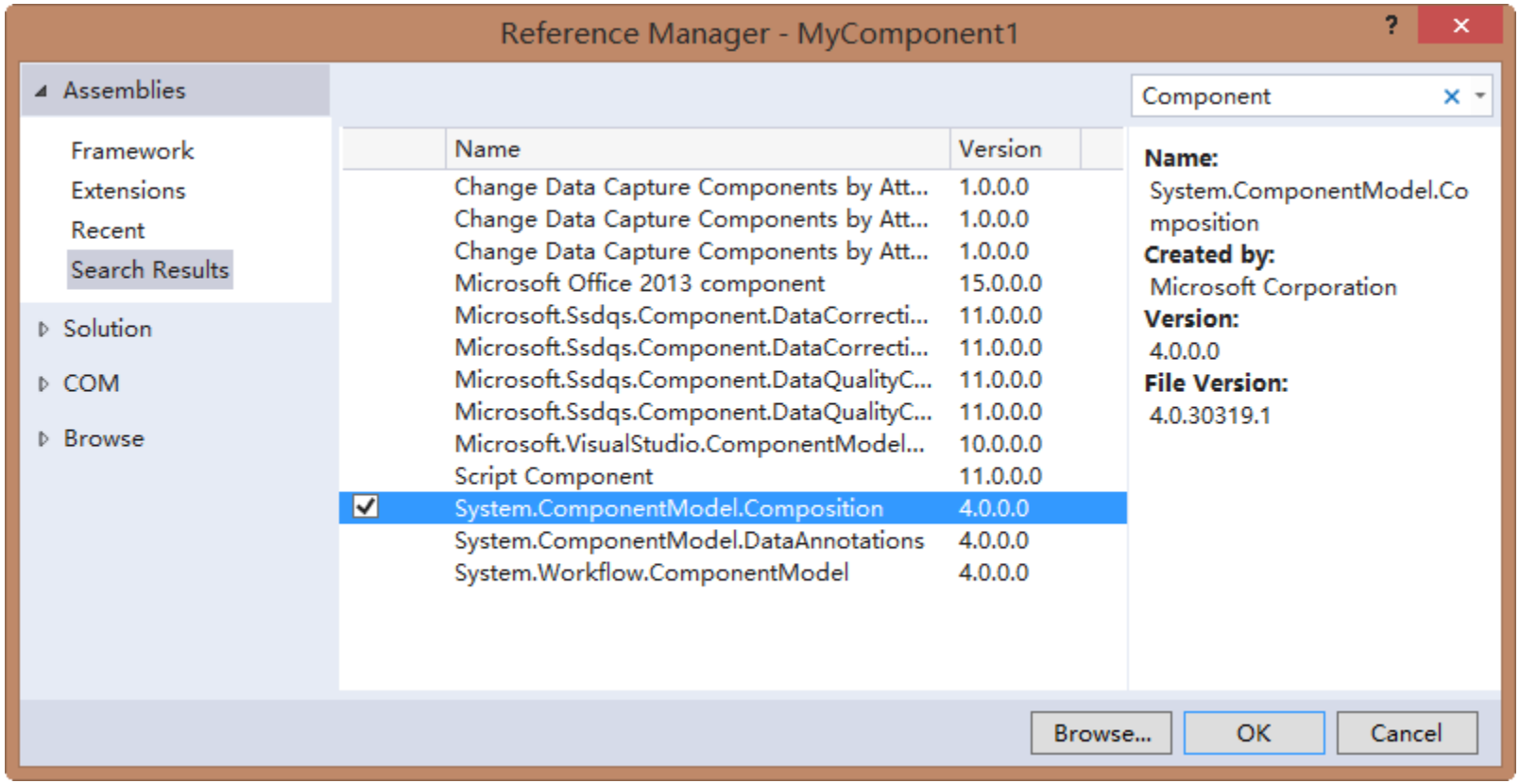
ASP.NET MVC插件化技术

北京理工大学计算机学院 金旭亮

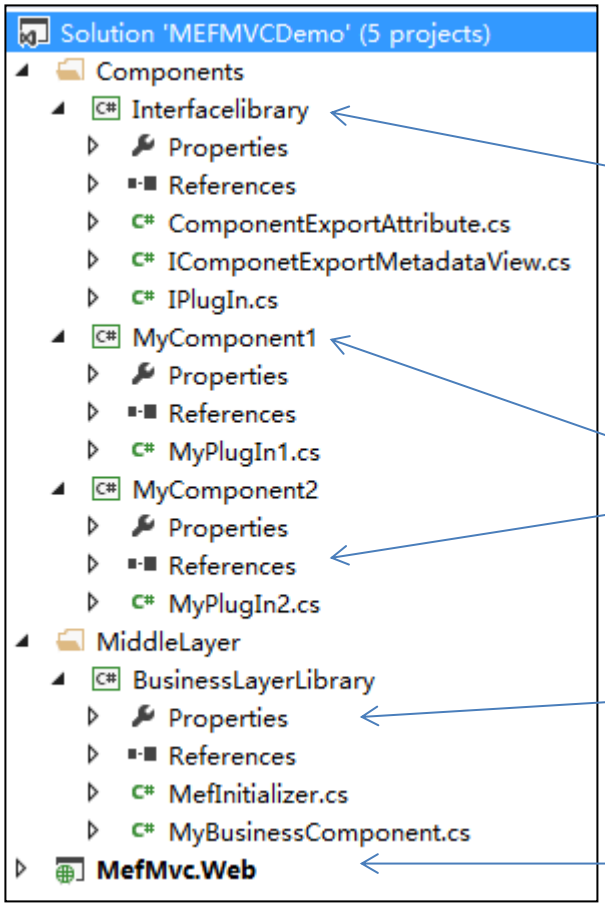
主要内容

- 示例项目简介
- 示例项目技术要点

- **MEF位于System.ComponetModel.Composition这个命名空间中**



示例解决方案



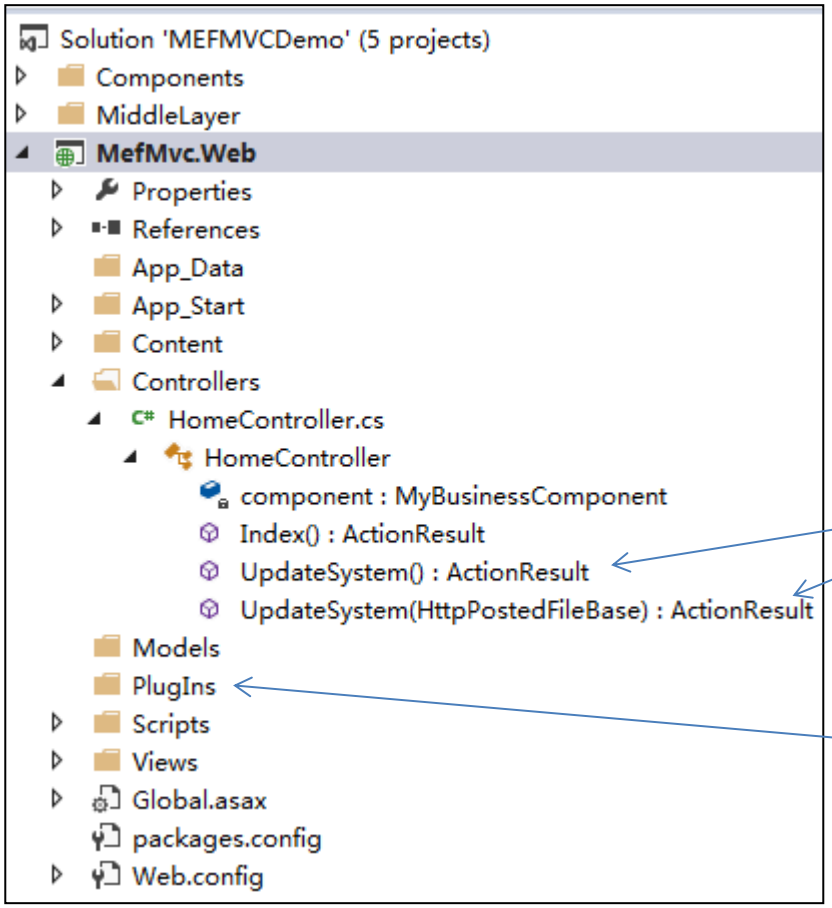
接口库，定义所有插件要实现的接口以及用于导出的自定义Attribute

插件项目，每个项目生成一个插件

业务逻辑层项目，其中的业务逻辑组件需要动态地组合相应插件，同时包容一个完成Mef容器初始化的类

表示层，一个ASP.NET MVC项目

表示层项目

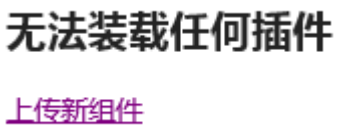


这两个方法用于动态地上传插件

用于保存上传插件的文件夹

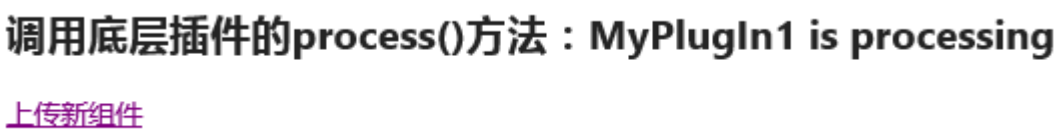
程序的运行截图-1

刚开始时没有任何插件.....



将组件1上传到plugIns文件夹下

组件1将被装载，其方法被调用



程序的运行截图-2

上传新组件

上传新的组件: Debug\MyComponent2.dll 浏览...

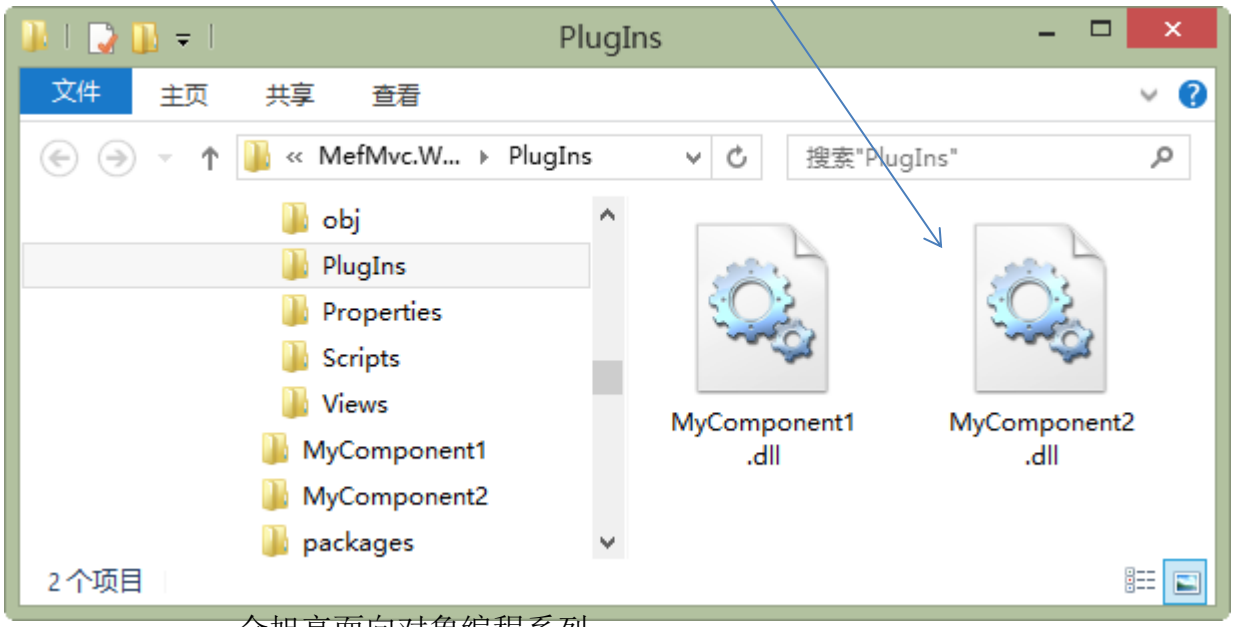
上传

上传组件2，版本比组件1要高

系统将使用版本高的组件2，而不是组件1

调用底层插件的process()方法：MyPlugIn2 is processing

上传新组件



示例项目技术要点

InterfaceLibrary项目

- 定义插件要实现的接口：

```
/// <summary>
/// 所有插件，都必须实现此接口
/// </summary>
7 references
public interface IPlugIn
{
    3 references
    string process();
}
```

- 定义插件导出元数据视图接口：

```
1 reference
public interface IComponentExportMetadataView
{
    2 references
    int VersionCode { get; }
}
```

InterfaceLibrary项目

- 自定义Attribute，简化指定插件版本的工作

```
[MetadataAttribute]
[AttributeUsage(AttributeTargets.Class,AllowMultiple=false)]
3 references
public class ComponentExportAttribute:ExportAttribute
{
    /// <summary>
    /// 本组件的版本号
    /// </summary>
    2 references
    public int VersionCode { get; set; }
    2 references
    public ComponentExportAttribute()
        : base(typeof(IPlugIn))
    {

    }
}
```

MyComponent1项目

- 每个插件应该放到独立的类库项目中，实现并导出插件接口，同时指定版本值：

```
[Export(typeof(IPlugIn))]
[PartCreationPolicy(CreationPolicy.NonShared)]
[ComponentExport(VersionCode = 1)]
0 references
public class MyPlugIn1 : IPlugIn
{
    3 references
    public string process()
    {
        return "MyPlugIn1 is processing";
    }
}
```

- 在Web项目中，建议把插件生存策略为NonShared，这就是说，为每个HTTP请求生成一个新的插件实例，否则，插件中的方法应该是线程安全的，这有可能会带来性能损失。

业务逻辑组件与插件

- 在真实的项目中，业务逻辑层通常包容多个项目，每个项目中都定义有多个业务逻辑组件。
- 在本示例中，仅定义了一个虚拟的业务类，它使用MEF动态地从网站的PlugIns文件夹中动态地组合一个版本最高的组件完成其业务处理流程。
- 这些插件通常用于实现经常变化的业务处理逻辑，比如某电子商务网站在双十一时价格特别优惠，双十一后恢复原价。这种不同的商品价格计算方法，就适合于以插件实现。

BusinessLayerLibrary项目

```
/// <summary>
/// 一个使用底层插件的虚拟业务类
/// </summary>
3 references
public class MyBusinessComponent
{
    /// <summary>
    /// 引用实际上使用的插件
    /// </summary>
    private IPlugIn _plugin=null;
    /// <summary>
    /// 保存所有在PlugIns文件夹下的插件引用
    /// </summary>
    [ImportMany(AllowRecomposition=true)]
    private IEnumerable<Lazy<IPlugIn,IComponentExportMetadataView>> plugIns;
    /// <summary>
    /// 在此动态组合插件，检查所有插件版本，选取版本最高的插件使用
    /// </summary>
    1 reference
    public MyBusinessComponent()...
    /// <summary>
    /// 启动某业务处理流程，使用_plugIn所引用的插件
    /// </summary>
    /// <returns></returns>
    1 reference
    public string BeginProcess()...
}
```

表示层的Web项目与业务逻辑层

- Web项目通常需要引用业务逻辑层中的组件完成各种功能，比较简单的方式是直接实例化业务逻辑层中的相应组件，在本示例中采用了这种方式。
- 在真实的项目中，还经常看到会在Web层与业务逻辑层中添加一个服务层，并且使用Ninject/AutoFac之类IoC容器降低Web层与业务逻辑层组件之间的耦合性。

MefMvc.Web

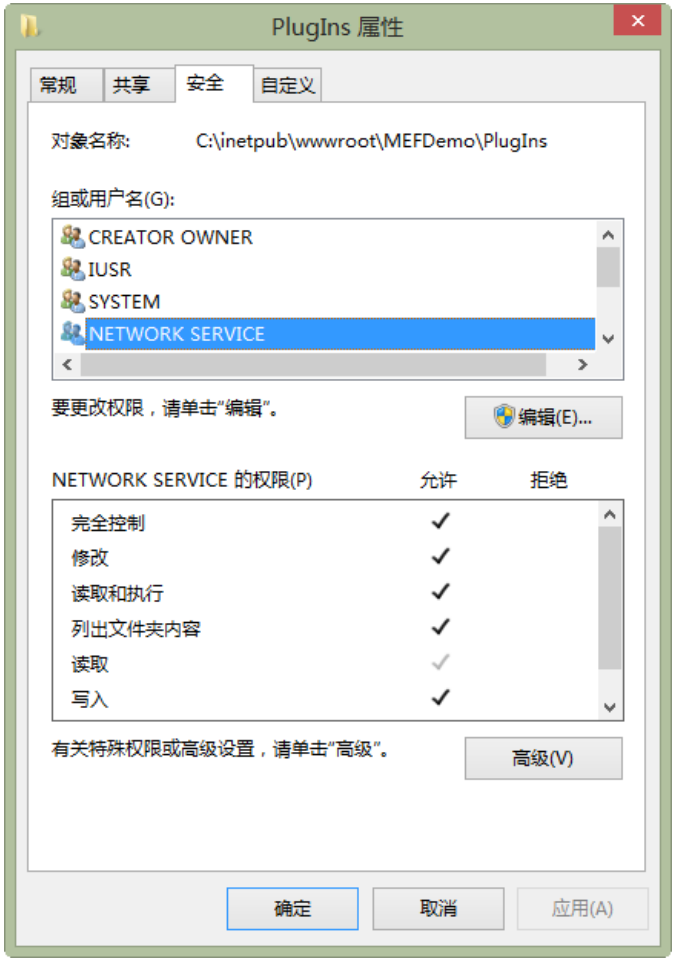
- 在MefMvc.Web项目中，通过网页直接上传插件dll到PlugIns文件，然后直接实例化业务逻辑层中的MyBusinessComponent类，在其构造函数中动态组合版本最高的组件。
- 在实际项目中，直接通过Web网页上传插件是比较危险的，因为它能直接修改系统的功能与业务流程。所以一定要经过身份验证后才允许上传插件（可以使用MVC内置的权限管理机制），给上传插件的Action方法添加[Authorize]属性指定用户名或角色名。

```
public class HomeController : Controller
{
    /// <summary>
    /// 实例化相应的业务逻辑组件
    /// </summary>
    private MyBusinessComponent component = new MyBusinessComponent();
    /// <summary>
    /// 启动执行业务流程
    /// </summary>
    /// <returns></returns>
    0 references
    public ActionResult Index()...
    /// <summary>
    /// 上传组件以更新系统
    /// </summary>
    /// <returns></returns>
    0 references
    public ActionResult UpdateSystem()...
    [HttpPost]
    0 references
    public ActionResult UpdateSystem(HttpPostedFileBase plugin)...

}
```

部署时的注意事项

- 当把MVC网站部署到IIS上时，要注意必须在资源管理器中为PlugIns文件夹设置正确的权限：



要给IIS使用的系统账号（本例中为Network Service）赋与PlugIns“完全控制”的权限，否则，上传的插件将无法保存。