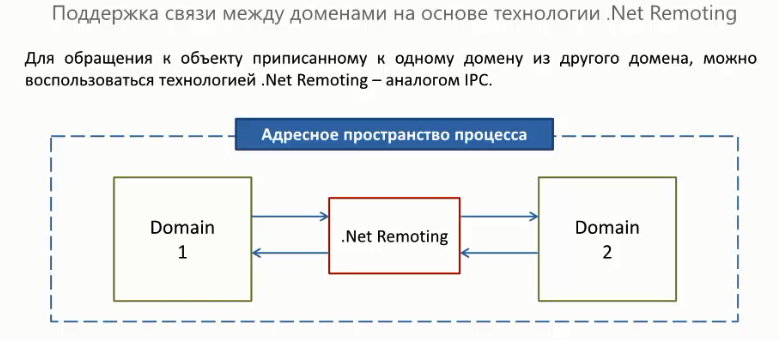


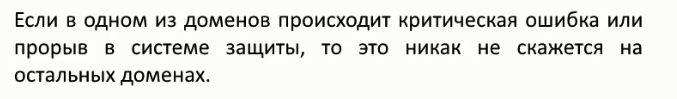
Mscorlib – разделяется между доменами



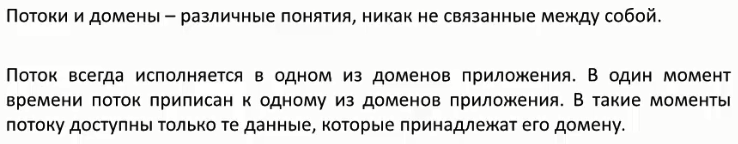
Междоменное взаимодействие намного быстрей чем межпроцессорное

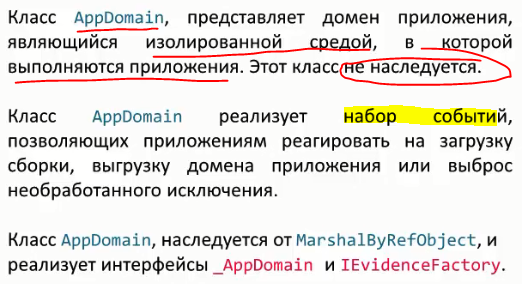
Assembly.Load(….) – загружается в домен приложения в связи с этим выгрузить в дальнейшем сборки не представиться возможным (очень накладно так как занимают память), поэтому следует загружать сборки во вторичный домен приложения, его впоследствии можно удалить вместе со сборками

Чужие сборки следует загружать во вторичные домены



Статические поля не разделяются между доменами и они будут для каждого уникальны





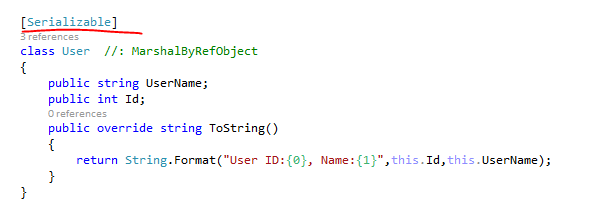
MarshalByRefObject

разрешает доступ к объектам через границы доменов приложения в приложениях, поддерживающих удаленное взаимодействие.

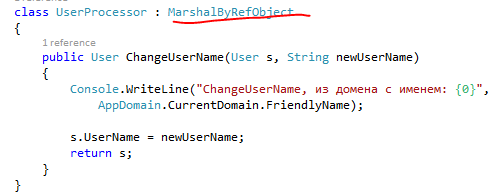
Если класс помечен атрибутом [Serializable] то он передается во вторичный домен по ЗНАЧЕНИЮ

namespace Domains

{



Если класс наследуется от класса MarshalByRefObject то он передается во вторичный домен по ССЫЛКЕ



// Создание второго домена приложения.

AppDomain secondDomain = AppDomain.CreateDomain("Second Domain");

// Создание объекта внутри второго домена. (передается имя сборки и полное квалифицированное имя класса)

ObjectHandle

Прокси обертка над объектом позволяющая работать с ним

ObjectHandle userHandle = secondDomain.CreateInstance("Domains", "Domains.UserProcessor");

Чтобы распаковать обертку прокси объекта

// Создание прозрачного прокси-переходника для взаимодействия с объектом из другого домена.

var userProxy = (UserProcessor)userHandle.Unwrap();

чтобы проверить что объект является прозрачным прокси переходником

// MyClass instance = (MyClass)secondDomain.CreateInstanceAndUnwrap("Domains", "Domains.MyClass");

// Проверка: Действительно ли прозрачный переходник предоставлен?

Console.WriteLine( RemotingServices.IsTransparentProxy(userProxy)

? "Развернутый объект является прокси-сервером."

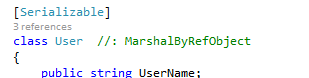
: "Развернутый объект не является прокси-сервером.");

// Вывод на экран текущего домена приложения.

Console.WriteLine("Метод Main находится в домене с именем: {0}",

AppDomain.CurrentDomain.FriendlyName);

1 и 2 разные, т.к. класс User передается по значению и происходит сериализация и десириализация объекта



CreateInstance – имя сборки и имя типа

AppDomainSetup ads1 = new AppDomainSetup() к этому примеру не относится, это если сборки в ином месте

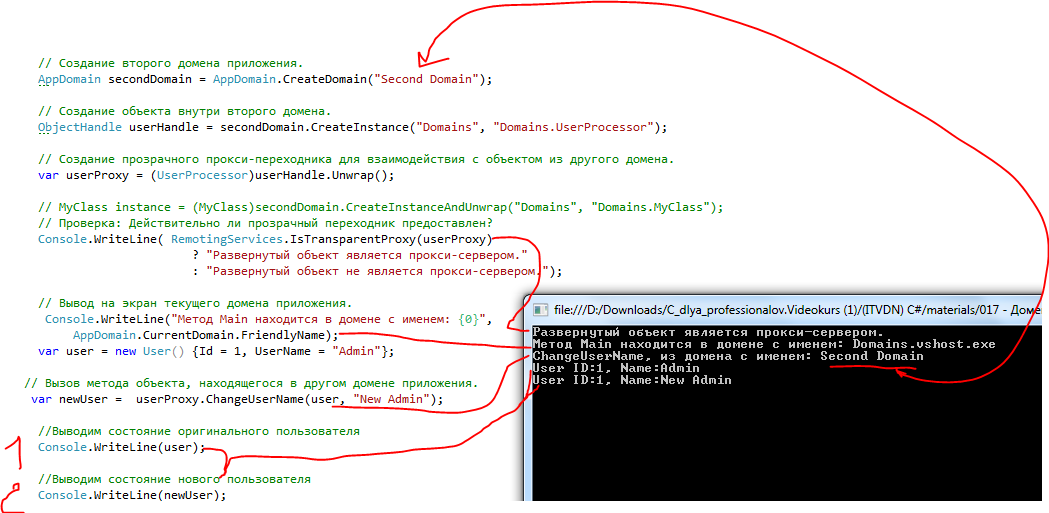
{

PrivateBinPath = pluginsPath – Полный путь к папке где лежат сборки

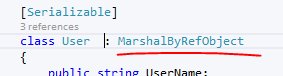
};

// Создание второго домена приложения.

AppDomain secondDomain = AppDomain.CreateDomain("Second Domain",null,ads1);



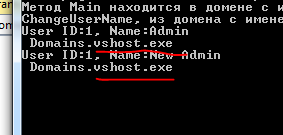
А если так



То



НО ОБА РАЗА ОБЪЕКТЫ СОЗДАЮТСЯ В ПЕРВИЧНОМ ДОМЕНЕ

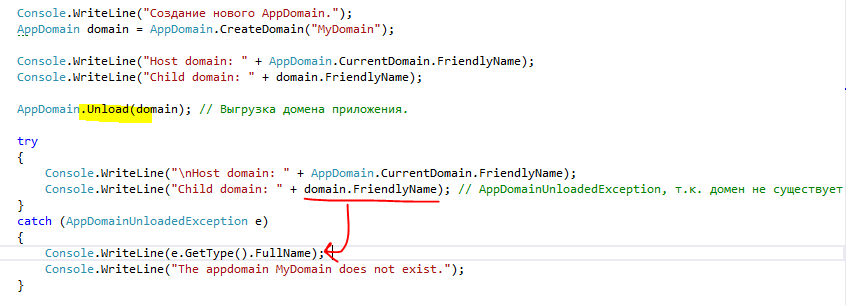


Чтобы класс созданный во вторичном домене вызывался в нем следует наследовать от MarshalByRefObject

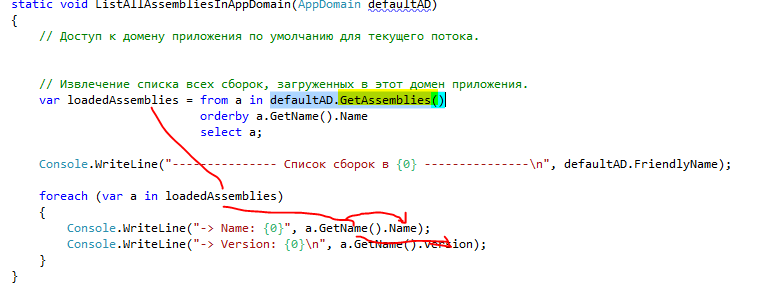


Выгрузка домена приложения

AppDomain.Unload(domain); // Выгрузка домена приложения.



defaultAD.GetAssemblies() – получение всех загруженных в домен сборок



// Возвращает базовый каталог, в котором распознаватель сборок производит поиск.

Console.WriteLine("\nApplication base is: " + domain.BaseDirectory); - месторасположение exe файла

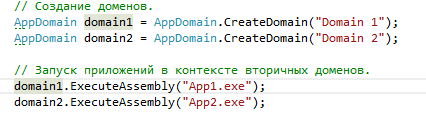
// Является ли домен используемым по умолчанию?

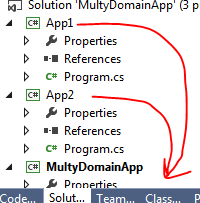
Console.WriteLine("\nIs {0} the default domain? - {1}", domain.FriendlyName, domain.IsDefaultAppDomain()); - главный домен? (который создан программой по умолчанию)

Запуск разных приложений в контексте одного процесса

Для выполнения этого проекта требуется скопировать исполняемые файлы App1.exe и App2.exe

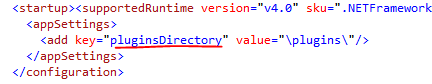
в папку с исполняемым файлом MultyDomainApp.exe





Подключение иных сборок

String pluginsPath = Path.GetDirectoryName(Assembly.GetEntryAssembly().Location) + ConfigurationManager.AppSettings["pluginsDirectory"]; – получение пути по которой находится сборка



String[] pluginLibraries = Directory.GetFiles(pluginsPath, "\*.dll"); - получение всех файлов которые подпадают под маску \*.dll

// Регистрируем обработчик загрузки необходимых типов в контекст рефлексии

AppDomain.CurrentDomain.ReflectionOnlyAssemblyResolve += ad2\_ReflectionOnlyAssemblyResolve; (метод обработки)

// Этот обработчик вызывается когда CLR пытается неудачно выполнить привязку сборки в контексте отражения.

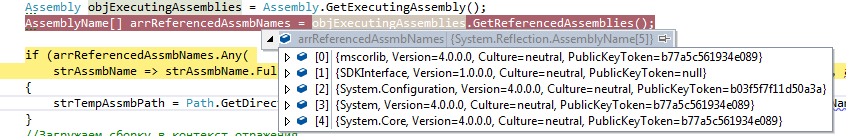
static Assembly ad2\_ReflectionOnlyAssemblyResolve(object sender, ResolveEventArgs args)

{

//Получаем список связанных сборок

string strTempAssmbPath = "";

Assembly objExecutingAssemblies = Assembly.GetExecutingAssembly(); - получение всех сборок



Метод подписанный на обработчик

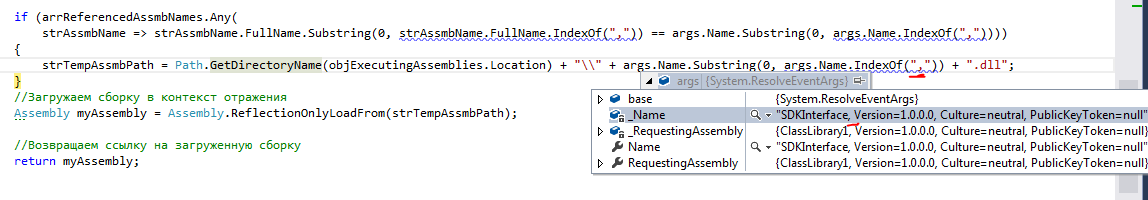
AssemblyName[] arrReferencedAssmbNames = objExecutingAssemblies.GetReferencedAssemblies();

if (arrReferencedAssmbNames.Any(

strAssmbName => strAssmbName.FullName.Substring(0, strAssmbName.FullName.IndexOf(",")) == args.Name.Substring(0, args.Name.IndexOf(","))))

{

strTempAssmbPath = Path.GetDirectoryName(objExecutingAssemblies.Location) + "\\" + args.Name.Substring(0, args.Name.IndexOf(",")) + ".dll";



}

//Загружаем сборку в контекст отражения

Assembly myAssembly = Assembly.ReflectionOnlyLoadFrom(strTempAssmbPath);

//Возвращаем ссылку на загруженную сборку

return myAssembly;

}

// Загружаем в контекст рефлексии сборки с плагинами (Код, загруженный в этот контекст, может быть проанализирован,

// но не может быть выполнен)

// Детальнее: http://msdn.microsoft.com/ru-ru/library/system.reflection.assembly.reflectiononlyloadfrom.aspx

foreach (Assembly assembly in pluginLibraries.Select(Assembly.ReflectionOnlyLoadFrom))

foreach (TypeInfo type in assembly.DefinedTypes)

{

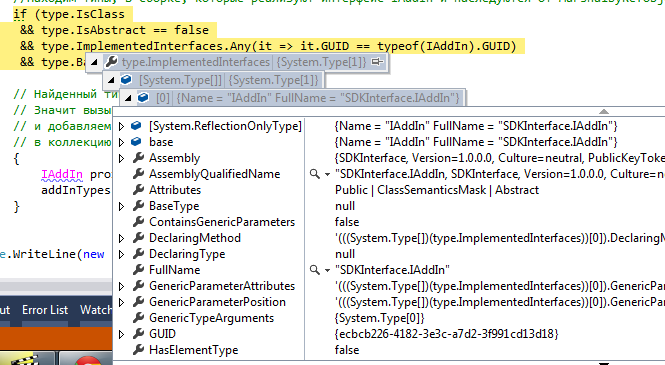
//Находим типы, в сборке, которые реализуют интерфейс IAddIn и наследуются от MarshalByRefObject

if (type.IsClass

&& type.IsAbstract == false

&& type.ImplementedInterfaces.Any(it => it.GUID == typeof(IAddIn).GUID)

&& type.BaseType == typeof(MarshalByRefObject)) – BaseType один т.к. наследуется только от одного класса



// Найденный тип - это подключаемый модуль

// Значит вызываем CreateInstanceFromAndUnwrap создавая экземпляр типа во вторичном домене,

// и добавляем ссылку на прокси-сервер

// в коллекцию addInTypes.

{

IAddIn proxy = ad2.CreateInstanceFromAndUnwrap(type.Assembly.CodeBase, type.FullName) as IAddIn;

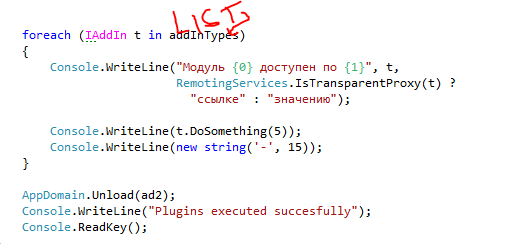


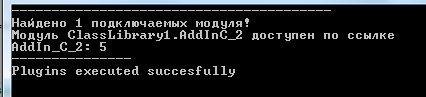
addInTypes.Add(proxy);

}

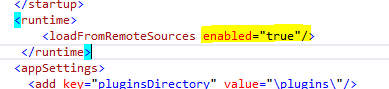
}

Выбираем нужные методы из всех сборок





Указывает, следует ли предоставлять полное доверие сборкам из удаленных источников.



Загрузка сборки

private void btn\_LoadAssemblies\_Click(object sender, System.EventArgs e)  
{  
    // Setup new application domain with configuration information  
    System.AppDomainSetup domaininfo = new System.AppDomainSetup();  
    domaininfo.ApplicationBase = AppDomain.CurrentDomain.BaseDirectory;  
  
    // Create a new application domain  
    AppDomain domain1 = AppDomain.CreateDomain( "MyTestAppDoman", null, domaininfo );  
  
    // Now load an assembly into the new app domain  
    System.Reflection.Assembly assembly = domain1.Load( @"C:\Projects\NET\_Framework\Projects\AppDomains\bin\Debug\MyAssembly.dll");  
  
    // Get the MyAssemblyClass and instantiate it  
    System.Type typClass = assembly.GetType( "MyAssemblyClass" );  
    Object obClass = System.Activator.CreateInstance( typClass );  
    typClass.GetMethod("foo").Invoke( obClass, null );  
  
    // Unload the app domain  
    AppDomain.Unload( domain1 );  
}



