LEARNING unity-container

#unity-container

Table of Contents

About

Chapter 1: Getting started with unity-container

Remarks

The Unity Container (Unity) is a lightweight, extensible dependency injection container. It facilitates building loosely coupled applications and provides developers with the following advantages: Simplified object creation, especially for hierarchical object structures and dependencies. [https://msdn.microsoft.com/en-us/library/ff647202.aspx]

It should also mention any large subjects within unity-container, and link out to the related topics. Since the Documentation for unity-container is new, you may need to create initial versions of those related topics.

Versions

Version	Release Notes	Release date
2.0.0	Unity 2	2011-05-05
2.1.0	Unity 2.1	2011-05-11
3.0.0	Unity 3	2013-04-26
3.5.0	Unity 3.5	2015-05-15
4.0.0	Unity handed over to OSS community	2015-10-06

Examples

Installation

In order to get started, you just need to install the **Unity** nuget package. Run the following command from the package manager console:

```
PM> Install-Package Unity
```

Alternatively, you can use Visual Studio to install Unity on a particular project using the *Manage NuGet Packages for Solution* option under Tools -> NuGet Package Manager.

Hello World

```
interface IGreeter
```

```
{
    void Greet();
}

class Greeter : IGreeter
{
    public void Greet()
    {
        Console.WriteLine("Hello World");
    }
}

class SpanishGreeter : IGreeter
{
    public void Greet()
    {
        Console.WriteLine("Hola Mundo");
    }
}

class FrenchGreeter : IGreeter
{
    public void Greet()
    {
        Console.WriteLine("Bonjour le Monde");
    }
}

class Program
{
    static void Main(string[] args)
    {
        var container = new UnityContainer()
            .RegisterType<IGreeter, SpanishGreeter>("spanish")
            .RegisterType<IGreeter, FrenchGreeter>("french")
            .RegisterType<IGreeter, Greeter>();

        //Get default registration. Outputs "Hello World"
        var greeter = container.Resolve<IGreeter>();
        greeter.Greet();

        //Get specific named registration. Outputs "Hola Mundo"
        greeter = container.Resolve<IGreeter>("spanish");
        greeter.Greet();

        //Get all named registrations (excludes the default one)
        //Outputs "Hola Mundo" and "Bonjour le Monde"
        foreach (var g in container.ResolveAll<IGreeter>())
        {
            g.Greet();
        }

        Console.ReadLine();
    }
}
```

Constructor Injection

```
interface IService
```

```
{
    void ProcessRequest();
}

interface IRepository
{
    IEnumerable<string> GetData();
}

class HelloWorldRepository : IRepository
{
    public IEnumerable<string> GetData()
    {
        return new[] { "Hello", "World" };
    }
}

class HelloWorldService : IService
{
    private readonly IRepository repo;
    public HelloWorldService(IRepository repo)
    {
        this.repo = repo;
    }
    public void ProcessRequest()
    {
        Console.WriteLine(String.Join(" ", this.repo.GetData()));
    }
}

class Program
{
    static void Main(string[] args)
    {
        var container = new UnityContainer()
            .RegisterType<IRepository, HelloWorldRepository>()
            .RegisterType<IService, HelloWorldService>();

        //Unity automatically resolves constructor parameters that knows about.
        //It will return a HelloWorldService with a HelloWorldRepository
        var greeter = container.Resolve<IService>();
        //Outputs "Hello World"
        greeter.ProcessRequest();

        Console.ReadLine();
    }
}
```

Read Getting started with unity-container online: https://riptutorial.com/unity-container/topic/5292/getting-started-with-unity-container

Chapter 2: Unity WebAPI

Examples

Setting up Unity with Web API.

1. Add Unity to your project.

If you use NuGet you can use the Unity-package. Run `Install-Package Unity` in Package Manager Console. This will add the Unity library (and it's dependencies) to your project.

2. Create an implementation of `IDependencyResolver`.

For example:

```
public class UnityResolver : IDependencyResolver
{
    protected IUnityContainer Container;

    public UnityResolver(IUnityContainer container)
    {
        if (container == null)
        {
            throw new ArgumentNullException("container");
        }
        this.Container = container;
    }

    public object GetService(Type serviceType)
    {
        try
        {
            return Container.Resolve(serviceType);
        }
        catch (ResolutionFailedException)
        {
            return null;
        }
    }

    public IEnumerable<object> GetServices(Type serviceType)
    {
        try
        {
            return Container.ResolveAll(serviceType);
        }
        catch (ResolutionFailedException)
        {
            return new List<object>();
        }
    }

    public IDependencyScope BeginScope()
    {
        var child = Container.CreateChildContainer();
```

```
            return new UnityResolver(child);
        }

        public void Dispose()
        {
            Container.Dispose();
        }
    }
```

3. Register your `IDependencyResolver` in your `WebApiConfig`.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Routes goes here..

        // Create your container.
        var container = new UnityContainer();

        // Do registrations here...

        // Assign your container.
        config.DependencyResolver = new UnityResolver(container);
    }
}
```

Read Unity WebAPI online: https://riptutorial.com/unity-container/topic/6396/unity-webapi

Credits

S. No	Chapters	Contributors
1	Getting started with unity-container	Clemens Tolboom, Community, Daniel J.G.
2	Unity WebAPI	smoksnes