WEIHONG

Dependency Injection in xunit

Amazing .NET

Microsoft® MVP Most Valuable Professional

Weihan Li

Intro

- 一个好的项目，它的背后有一系列的测试用例
- 测试覆盖率是一个高质量项目的重要指标
- 完善的测试间接关系着产品的质量以及用户的满意度和信任

- 依赖注入已经是现代化应用的标配，.NET Core 从一开始就集成了依赖注入
- 对于测试项目，依赖注入也不能缺席，依赖注入可以使得测试项目更容易
- Xunit.DependencyInjection 是基于微软的 GenericHost 来实现的 xunit 扩展，能够让你更方便的在测试项目中使用依赖注入，可以更好的和 .NET 集成和扩展

How to in xunit

```
public class DatabaseFixture : IDisposable
{
    public DatabaseFixture()
    {
        Db = new SqlConnection("MyConnectionString");

        // ... initialize data in the test database ...
    }

    public void Dispose()
    {
        // ... clean up test data from the database ...
    }

    public SqlConnection Db { get; private set; }
}

public class MyDatabaseTests : IClassFixture<DatabaseFixture>
{
    DatabaseFixture fixture;

    public MyDatabaseTests(DatabaseFixture fixture)
    {
        this.fixture = fixture;
    }

    // ... write tests, using fixture.Db to get access to the SQL Server ...
}
```

How to in xunit

```
public class DatabaseFixture : IDisposable
{
    public DatabaseFixture()
    {
        Db = new SqlConnection("MyConnectionString");

        // ... initialize data in the test database ...
    }

    public void Dispose()
    {
        // ... clean up test data from the database ...
    }

    public SqlConnection Db { get; private set; }
}

[CollectionDefinition("Database collection")]
public class DatabaseCollection : ICollectionFixture<DatabaseFixture>
{
    // This class has no code, and is never created. Its purpose is simply
    // to be the place to apply [CollectionDefinition] and all the
    // ICollectionFixture<> interfaces.
}
```

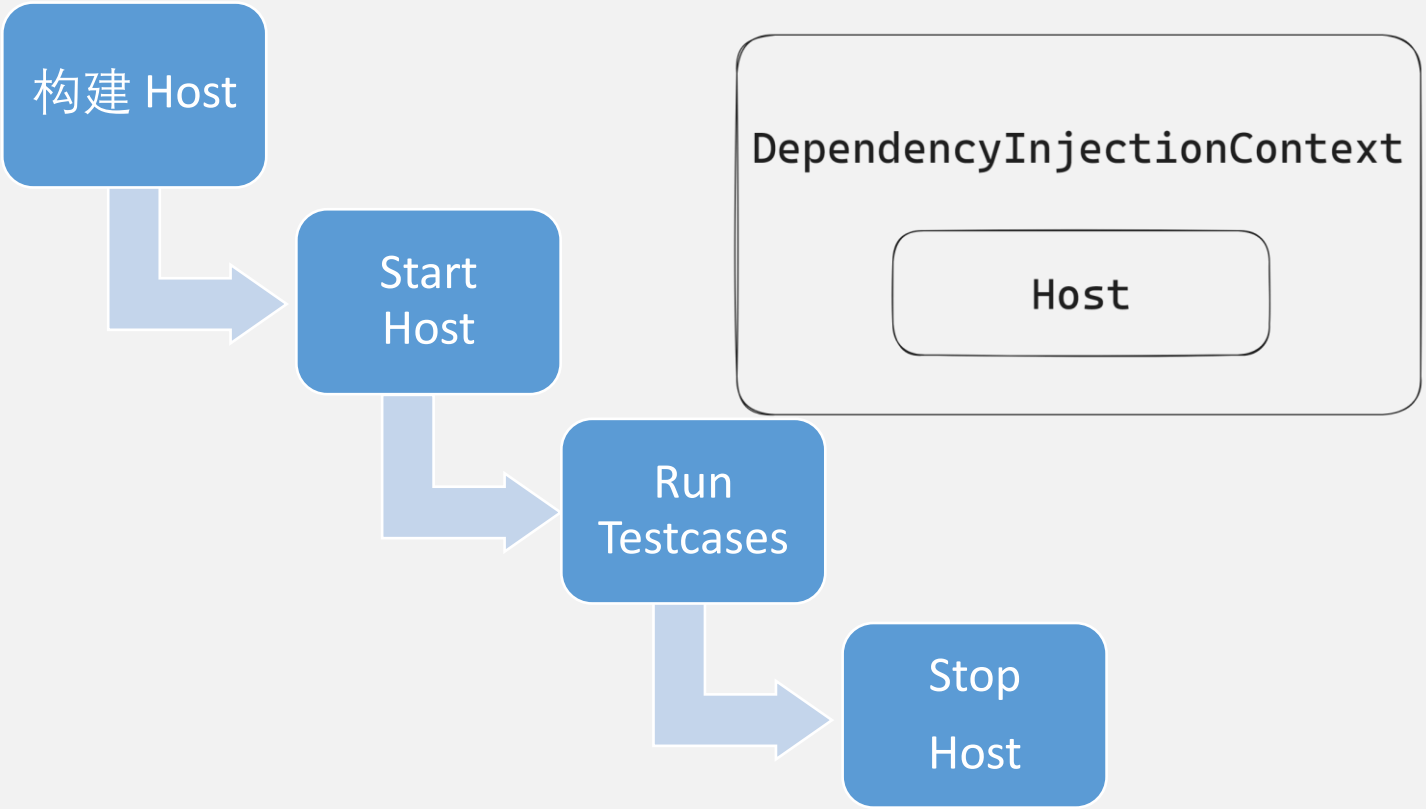
```
[Collection("Database collection")]
public class DatabaseTestClass1
{
    DatabaseFixture fixture;

    public DatabaseTestClass1(DatabaseFixture fixture)
    {
        this.fixture = fixture;
    }
}

[Collection("Database collection")]
public class DatabaseTestClass2
{
    // ...
}
```

With Xunit.DependencyInjection

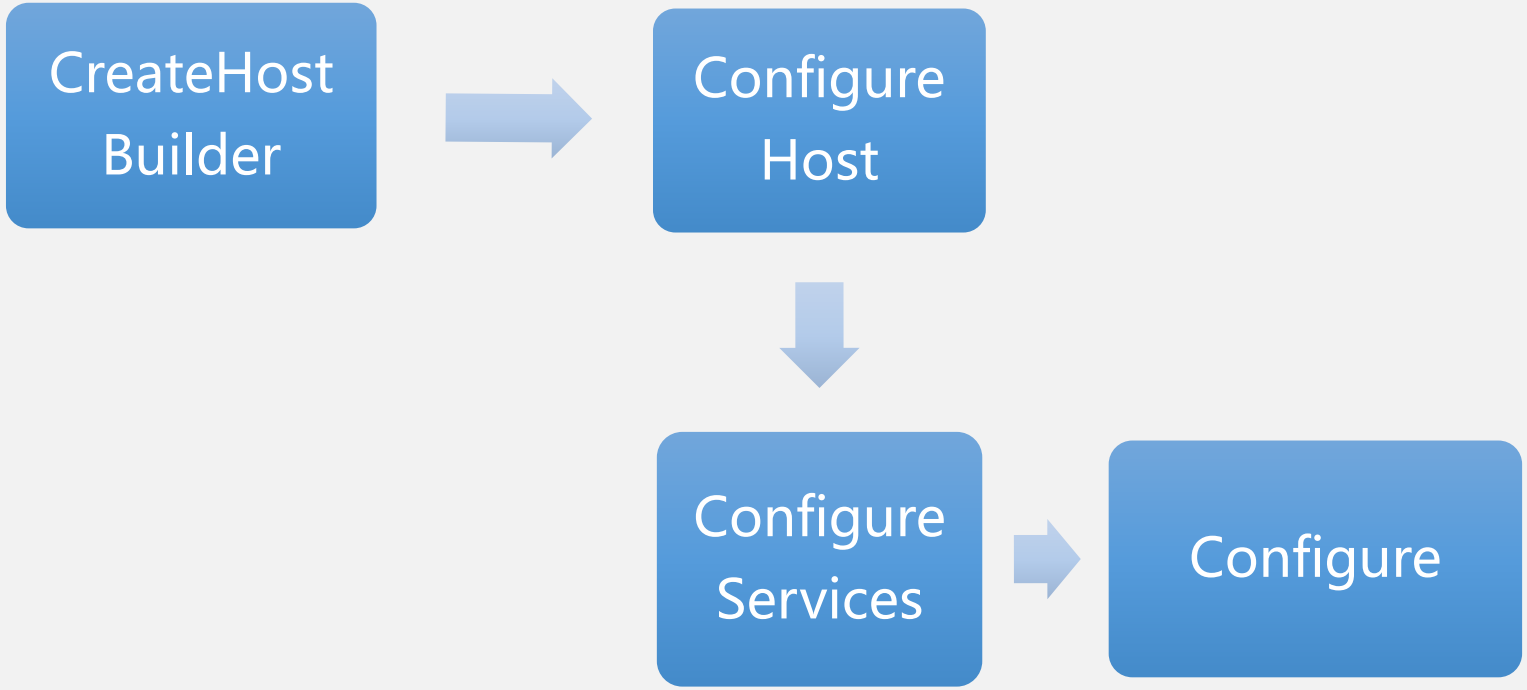
```
public class IdGeneratorTest
{
    private readonly IIdGenerator _idGenerator;

    public IdGeneratorTest(IIdGenerator idGenerator)
    {
        _idGenerator = idGenerator;
    }

    [Fact]
    public void NewIdTest()
    {
        var newId = _idGenerator.NewId();
        Assert.NotNull(newId);
        Assert.NotEmpty(newId);
    }
}
```

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IIdGenerator, GuidIdGenerator>();
    }
}
```

Workflow


构建 Host
Start Host
Run Testcases
Stop Host
DependencyInjectionContext
Host

Host 构建流程


CreateHost Builder
Configure Host
Configure Services
Configure

Host 构建流程

在测试项目中创建一个 Startup 类来控制 Host 的构建

```
public class Startup
{
    // create custom hostBuilder with this method
    // public IHostBuilder CreateHostBuilder()
    // {
    // }

    // custom host build
    public void ConfigureHost(IHostBuilder hostBuilder)
    {
        // hostBuilder
        //     .ConfigureHostConfiguration(builder =>
        //     {
        //         builder.AddInMemoryCollection(new Dictionary<string, string>()
        //             {
        //                 { "UserName", "Alice" }
        //             });
        //     });
    }

    // add services need to injection
    // ConfigureServices(IServiceCollection services)
    // ConfigureServices(IServiceCollection services, HostBuilderContext hostBuilderContext)
    // ConfigureServices(HostBuilderContext hostBuilderContext, IServiceCollection services)
    public void ConfigureServices(IServiceCollection services, HostBuilderContext hostBuilderContext)
    {
        // get configuration by hostBuilderContext.Configuration
        // eg: hostBuilderContext.Configuration["Environment"]

        //services.AddSingleton<CustomService>();
    }

    // Initialize logic, executed before running test cases
    public void Configure(IServiceProvider applicationServices)
    {
    }
}
```

StartupAttribute

```
[Startup(typeof(Startup1))]
public class StartupAttributeSample
{
    private readonly ICalc _calc;

    public StartupAttributeSample(ICalc calc)
    {
        _calc = calc;
    }

    [Fact]
    public void Calc_Test()
    {
        Assert.IsType<AddCalc>(_calc);
        Assert.Equal(2, _calc.Calc(1, 1));
    }
}
```

```
public interface ICalc
{
    int Calc(int num1, int num2);
}
file sealed class AddCalc : ICalc
{
    public int Calc(int num1, int num2) => num1 + num2;
}

file sealed class Startup1
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<ICalc, AddCalc>();
    }
}
```

StartupAttribute

```
[Startup(typeof(Startup2))]
public class StartupAttributeSample2
{
    private readonly ICalc _calc;
    private readonly ITestOutputHelper _outputHelper;

    public StartupAttributeSample2(ICalc calc, ITestOutputHelper outputHelper)
    {
        _calc = calc;
        _outputHelper = outputHelper;
    }

    [Fact]
    public void Calc_Test()
    {
        _outputHelper.WriteLine($"{nameof(_calc)}: {_calc.GetHashCode()}");
        Assert.IsType<MultiCalc>(_calc);
        Assert.Equal(1, _calc.Calc(1, 1));
    }


    [Fact]
    public void Calc_Test2()
    {
        _outputHelper.WriteLine($"{nameof(_calc)}: {_calc.GetHashCode()}");
        Assert.IsType<MultiCalc>(_calc);
        Assert.Equal(2, _calc.Calc(1, 2));
    }
}
```

```
[Startup(typeof(Startup2), Shared = false)]
public class StartupAttributeSample3
{
    private readonly ICalc _calc;
    private readonly ITestOutputHelper _outputHelper;

    public StartupAttributeSample3(ICalc calc, ITestOutputHelper outputHelper)
    {
        _calc = calc;
        _outputHelper = outputHelper;
    }

    [Fact]
    public void Calc_Test()
    {
        _outputHelper.WriteLine($"{nameof(_calc)}: {_calc.GetHashCode()}");
        Assert.IsType<MultiCalc>(_calc);
        Assert.Equal(1, _calc.Calc(1, 1));
        Assert.Equal(1, ((MultiCalc)_calc).Counter);
    }
}
```

```
Calc_Test [0 ms]
_calc: 23072233
```

```
Calc_Test2 [119 ms]
_calc: 23072233
```

```
Calc_Test [102 ms]
_calc: 53387259
```

Nested startup

```
public class NestedStartupSample
{
    private readonly IRandom _random;

    public NestedStartupSample(IRandom random)
    {
        _random = random;
    }

    [Fact]
    public void RandomTest()
    {
        var value = _random.GetValue(2);
        Assert.True(value is >= 0 and <= 2);
    }

    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddSingleton<IRandom, RandomService>();
        }
    }
}
```

GetStarted

```
public class Startup
{
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<DelayService>();
    }

    // Initialize logic
    0 references
    public void Configure(DelayService delayService)
    {
        while (!delayService.Ready())
        {
            Thread.Sleep(200);
        }
    }
}
```

```
public class Startup
{
    // create custom hostBuilder with this method
    //public IHostBuilder CreateHostBuilder()
    //{        You, a day ago • add GetStarted samples
    //    return Host.CreateDefaultBuilder();
    //}

    // custom host build
    0 references
    public void ConfigureHost(IHostBuilder hostBuilder)
    {
        hostBuilder
            .ConfigureHostConfiguration(builder =>
            {
                builder.AddJsonFile("appsettings.json", true);
            });
    }

    // add services need to injection
    // ConfigureServices(IServiceCollection services)
    // ConfigureServices(IServiceCollection services, HostBuilderContext hostBuilderContext)
    // ConfigureServices(HostBuilderContext hostBuilderContext, IServiceCollection services)
    0 references
    public void ConfigureServices(IServiceCollection services, HostBuilderContext hostBuilderContext)
    {
        var configuration = hostBuilderContext.Configuration;
        if ("Guid".Equals(configuration["AppSettings:IdType"], StringComparison.OrdinalIgnoreCase))
        {
            services.AddSingleton<IIdGenerator, GuidIdGenerator>();
        }
        else
        {
            services.AddSingleton<IIdGenerator, IntIdGenerator>();
        }
    }
}
```

GetStarted

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IIdGenerator, GuidIdGenerator>();
    }
}
```

```
public interface IIdGenerator
{
    string NewId();
}

public class GuidIdGenerator : IIdGenerator
{
    public string NewId()
    {
        return Guid.NewGuid().ToString("N");
    }
}
```

```
public class IdGeneratorTest
{
  private readonly IIdGenerator _idGenerator;

  public IdGeneratorTest(IIdGenerator idGenerator)
  {
      _idGenerator = idGenerator;
  }

  [Fact]
  public void NewIdTest()
  {
      var newId = _idGenerator.NewId();
      Assert.NotNull(newId);
      Assert.NotEmpty(newId);
  }
```

```
[Theory]
[InlineData(null, "test")]
public void MethodInjectionTest([FromServices] IIdGenerator idGenerator, string data)
{
    Assert.NotNull(data);
    Assert.NotNull(idGenerator);
    Assert.Equal(_idGenerator, idGenerator);
    var newId = idGenerator.NewId();
    Assert.NotNull(newId);
    Assert.NotEmpty(newId);
}
```

GetStarted

```
[Theory]
[MemberData(nameof(GetTestData))]
public void MemberDataMethodInjectionTest([FromServices] IIdGenerator idGenerator, string data)
{
    Assert.NotNull(data);
    Assert.NotNull(idGenerator);
    Assert.Equal(_idGenerator, idGenerator);
    var newId = idGenerator.NewId();
    Assert.NotNull(newId);
    Assert.NotEmpty(newId);
}

public static IEnumerable<object?[]> GetTestData()
{
    yield return new object?[] { null, "test" };
}
```

```
public static class TestClass
{
    public static IEnumerable<object?[]> GetTestData(string data)
    {
        yield return new object?[] { null, data };
    }
}
```

```
[Theory]
[MethodData(nameof(GetTestData))]
public void MethodDataMethodInjectionTest([FromServices] IIdGenerator idGenerator, string data)
{
    Assert.NotNull(data);
    Assert.Equal("test", data);

    Assert.NotNull(idGenerator);
    Assert.Equal(_idGenerator, idGenerator);
    var newId = idGenerator.NewId();
    Assert.NotNull(newId);
    Assert.NotEmpty(newId);
}

[Theory]
[MethodData(nameof(TestClass.GetTestData), typeof(TestClass), "test2")]
public void MethodDataMethodInjectionExternalClassTest([FromServices] IIdGenerator idGenerator, string data)
{
    Assert.NotNull(data);
    Assert.Equal("test2", data);

    Assert.NotNull(idGenerator);
    Assert.Equal(_idGenerator, idGenerator);
    var newId = idGenerator.NewId();
    Assert.NotNull(newId);
    Assert.NotEmpty(newId);
}
```

Autofac Integration

```
public class Startup
{
    // custom host build
    public void ConfigureHost(IHostBuilder hostBuilder)
    {
        hostBuilder.UseServiceProviderFactory(new AutofacServiceProviderFactory(builder =>
        {
            builder.RegisterType<GuidIdGenerator>()
                .As<IIdGenerator>()
                .SingleInstance()
                ;
        }));
    }
}
```

AspectCore Integration

```
public class Startup
{
    // custom host build
    public void ConfigureHost(IHostBuilder hostBuilder)
    {
        hostBuilder
            .UseServiceProviderFactory(new DynamicProxyServiceProviderFactory())
            ;
    }

    // add services need to injection
    // ConfigureServices(IServiceCollection services)
    // ConfigureServices(IServiceCollection services, HostBuilderContext hostBuilderContext)
    // ConfigureServices(HostBuilderContext hostBuilderContext, IServiceCollection services)
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IIdGenerator, GuidIdGenerator>();

        services.ConfigureDynamicProxy(config =>
        {
            config.Interceptors.AddTyped<CounterInterceptor>(Predicates.ForService(nameof(IIdGenerator)));
        });
    }
}
```

TestServer Integration

```
public class Startup
{
    // custom host build
    public void ConfigureHost(IHostBuilder hostBuilder)
    {
        hostBuilder
            .ConfigureWebHostDefaults(builder =>
            {
                builder.UseStartup<TestWebApp.Startup>();

                builder.UseTestServer();
                builder.ConfigureServices(services =>
                {
                    services.AddSingleton(sp => sp.GetRequiredService<IHost>()
                        .GetTestClient()
                    );
                });
            })
            ;
    }
}
```

```
public class ApiTest
{
    private readonly HttpClient _httpClient;

    public ApiTest(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    [Fact]
    public async Task GetTest()
    {
        var response = await _httpClient.GetAsync("api/test");
        Assert.True(response.IsSuccessStatusCode);

        var responseText = await response.Content.ReadAsStringAsync();
        Assert.NotEmpty(responseText);

        var result = JsonSerializer.Deserialize<Result<bool>>(responseText, new JsonSerializerOptions()
        {
            PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
        });
        Assert.NotNull(result);
        Assert.True(result.Data);
    }
}
```

Minimal API Testing

```
<ItemGroup>
  <PackageReference Include="Xunit.DependencyInjection.AspNetCoreTesting" />
</ItemGroup>
```

```
public sealed class Startup
{
    public IHostBuilder CreateHostBuilder() =>
        MinimalApiHostBuilderFactory.GetHostBuilder<Program>();
}
```

```
public class ApiTest
{
    private readonly HttpClient _testClient;
    private readonly IRandomService _randomService;

    public ApiTest(HttpClient testClient, IRandomService randomService)
    {
        _testClient = testClient;
        _randomService = randomService;
    }

    [Fact]
    public async Task HelloTest()
    {
        var responseText = await _testClient.GetStringAsync("/");
        Assert.Equal("Hello MinimalAPI", responseText);
    }

    [Fact]
    public void ServiceTest()
    {
        var num = _randomService.GetNumber();
        Assert.True(num < 100);
    }
}
```

Hosted Service

```
[Route("ready")]
public IActionResult Ready([FromServices] ReadyChecker readChecker)
{
    if (readChecker.Check())
    {
        return Ok();
    }
    return BadRequest();
}
```

```
public class ReadyChecker
{
    private static int _readyStatus;

    static ReadyChecker()
    {
        Task.Run(async () =>
        {
            await Task.Delay(3000);
            _readyStatus = 1;
        });
    }

    public bool Check()
    {
        return _readyStatus > 0;
    }
}
```

```
public class ReadyCheckHostedService : IHostedService
{
    2 references
    private readonly IServiceProvider _serviceProvider;
    3 references
    private readonly ILogger<ReadyCheckHostedService> _logger;

    0 references
    public ReadyCheckHostedService(IServiceProvider serviceProvider, ILogger<ReadyCheckHostedService> logger)
    {
        _serviceProvider = serviceProvider;
        _logger = logger;
    }

    0 references
    public async Task StartAsync(CancellationToken cancellationToken)
    {
        var client = _serviceProvider.GetRequiredService<HttpClient>();
        while (true)
        {
            using var response = await client.GetAsync("api/ready", cancellationToken);
            if (response.IsSuccessStatusCode)
            {
                break;
            }
            _logger.LogWarning("API has not ready");
            await Task.Delay(1000, cancellationToken);
        }
        _logger.LogInformation("API has ready");
    }

    0 references
    public Task StopAsync(CancellationToken cancellationToken)
    {
        return Task.CompletedTask;
    }
}
```

TestOutputHelperAccessor

```
public class InvokeHelper
{
    private readonly ITestOutputHelperAccessor _outputHelperAccessor;

    public InvokeHelper(ITestOutputHelperAccessor outputHelperAccessor)
    {
        _outputHelperAccessor = outputHelperAccessor;
    }

    public void Profile(Action action, string actionName)
    {
        var watch = Stopwatch.StartNew();
        action();
        watch.Stop();
        _outputHelperAccessor.Output?.WriteLine($"{actionName} elapsed:{watch.ElapsedMilliseconds}ms");
    }
}
```

```
public class HostedTest
{
    private readonly InvokeHelper _invokeHelper;

    public HostedTest(InvokeHelper invokeHelper)
    {
        _invokeHelper = invokeHelper;
    }

    [Fact]
    public void OutputHelperAccessorTest()
    {
        _invokeHelper.Profile(() =>
        {
            Thread.Sleep(3000);
        }, nameof(OutputHelperAccessorTest));
    }
}
```

OutputHelperAccessorTest passed

```
OutputHelperAccessorTest elapsed:3005ms
```

Logging

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddLogging(loggingBuilder => loggingBuilder.AddXunitOutput());
}
```

```
[Fact]
public void LoggingTest()
{
    _logger.LogDebug("Debug");
    _logger.LogInformation("Info");
    _logger.LogWarning("Warn");
    _logger.LogError("Error");
    _logger.LogCritical("Critical");
}
```

```
✔ LoggingTest [10 ms]
[2023-10-27 06:36:37Z] info: MoreFeatures.FeatureTest[0]
      Info
[2023-10-27 06:36:37Z] warn: MoreFeatures.FeatureTest[0]
      Warn
[2023-10-27 06:36:37Z] fail: MoreFeatures.FeatureTest[0]
      Error
[2023-10-27 06:36:37Z] crit: MoreFeatures.FeatureTest[0]
      Critical
```

Project Template

安装项目模板：dotnet new install Xunit.DependencyInjection.Template
创建项目：dotnet new create xunit-di -n TestProject && cd TestProject && ls

```
PS C:\projects\test\ConsoleApp1> dotnet new create xunit-di -n TestProject && cd TestProject && ls
The template "Xunit DependencyInjection Template" was created successfully.


    Directory: C:\projects\test\ConsoleApp1\TestProject

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          10/27/2023    20:28            460 ConfigurationTest.cs
-a---          10/27/2023    20:28           1557 Startup.cs
-a---          10/27/2023    20:28            905 TestProject.csproj

PS C:\projects\test\ConsoleApp1\TestProject> cat .\TestProject.csproj
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
```

Project Template

安装项目模板：dotnet new install Xunit.DependencyInjection.Template
创建项目: mkdir TestProject2 && cd TestProject2 && dotnet new create xunit-di -f net8.0

```
PS C:\projects\test\ConsoleApp1> mkdir TestProject2 && cd TestProject2 && dotnet new create xunit-di -f net8.0

    Directory: C:\projects\test\ConsoleApp1

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          10/27/2023    20:30                TestProject2
The template "Xunit DependencyInjection Template" was created successfully.


PS C:\projects\test\ConsoleApp1\TestProject2> ls

    Directory: C:\projects\test\ConsoleApp1\TestProject2

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          10/27/2023    20:30            461 ConfigurationTest.cs
-a---          10/27/2023    20:30           1558 Startup.cs
-a---          10/27/2023    20:30            905 TestProject2.csproj

PS C:\projects\test\ConsoleApp1\TestProject2> cat .\TestProject2.csproj
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
```

References && More

- https://github.com/pengweiqhca/Xunit.DependencyInjection
- https://github.com/WeihanLi/XunitDependencyInjection.Samples
- https://www.nuget.org/packages/Xunit.DependencyInjection
- https://www.nuget.org/packages/Microsoft.AspNetCore.TestHost
- https://www.nuget.org/packages/Xunit.DependencyInjection.Logging
- https://www.nuget.org/packages/Xunit.DependencyInjection.Template
- https://github.com/xunit/xunit
- https://xunit.net/docs/shared-context

提问答疑

Keep Testing

THANKS FOR WATCHING