عنوان: **آزمون واحد در MVVM به کمک تزریق وابستگی**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۴ ۱۷:۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM, Unit testing, Dependency Injection

یکی از خوبی‌های استفاده از Presentation Patternها بالا بردن تست پذیری برنامه و در نتیجه نگهداری کد می‌باشد.
MVVM الگوی محبوب برنامه نویسان WPF و Silverlight می‌باشد. به صرف استفاده از الگوی MVVM نمی‌توان اطمینان داشت که ViewModel کاملا تست پذیری داریم. به عنوان مثلا اگر در ViewModel خود مستقیما DialogBox کنیم یا ارجاعی از View دیگری داشته باشیم نوشتن آزمون‌های واحد تقریبا غیر ممکن می‌شود. قبلا درباره‌ی این مشکلات و راه حل آن مطلب در سایت منتشر شده است :

- MVVM و نمایش دیالوگ‌ها

در این مطلب قصد داریم سناریویی را بررسی کنیم که ViewModel از Background Worker جهت انجام عملیات مانند دریافت داده‌ها استفاده می‌کند.

Background Worker کمک می‌کند تا اعمال طولانی در یک Thread دیگر اجرا شود در نتیجه رابط کاربری Freeze نمی‌شود.

به این مثال ساده توجه کنید :

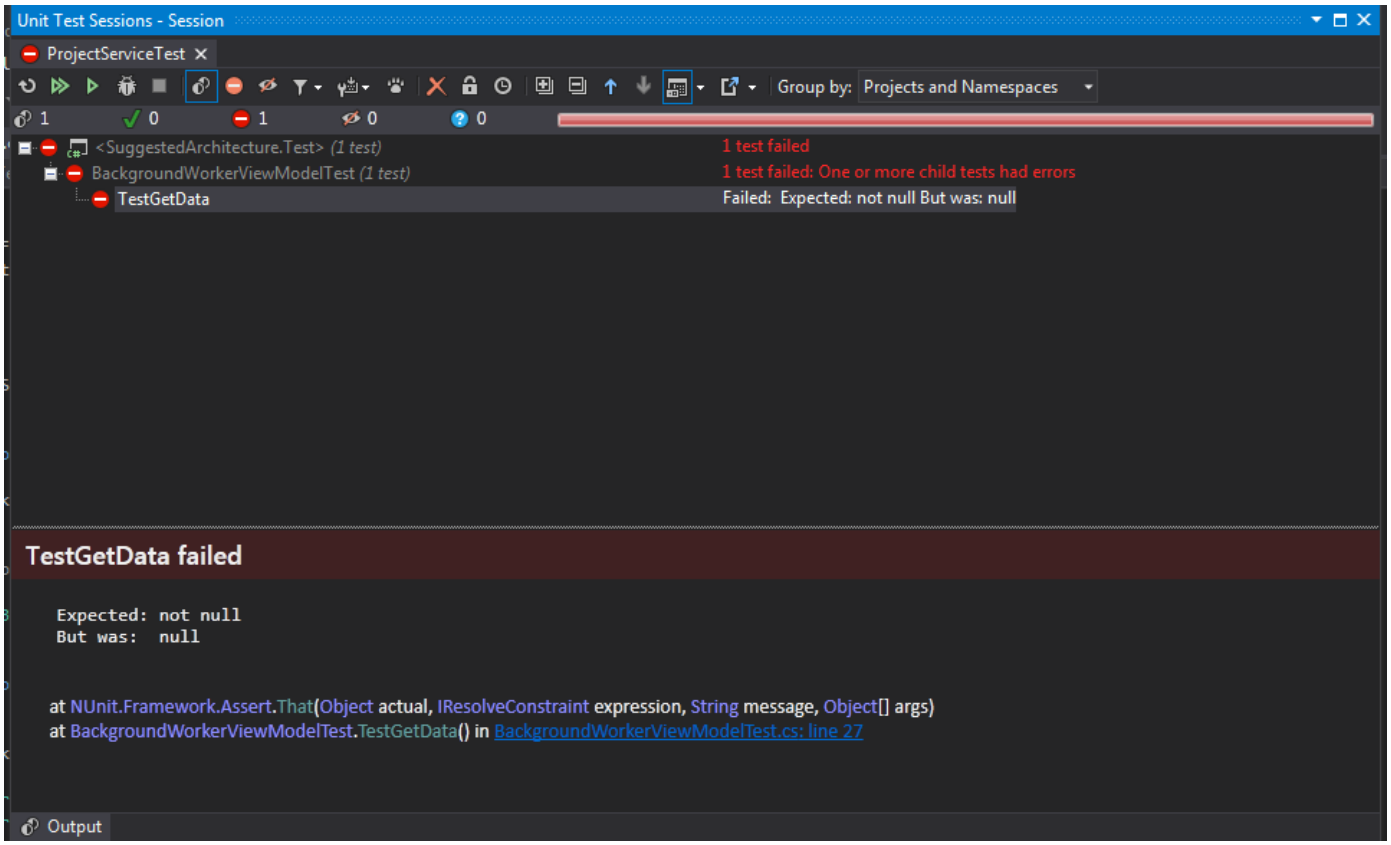
```
public class BackgroundWorkerViewModel : BaseViewModel
    {
        private List<string> _myData;

        public BackgroundWorkerViewModel()
        {
            LoadDataCommand = new RelayCommand(OnLoadData);
        }

        public RelayCommand LoadDataCommand { get; set; }

        public List<string> MyData
        {
            get { return _myData; }
            set
            {
                _myData = value;
                RaisePropertyChanged(() => MyData);
            }
        }

        public bool IsBusy { get; set; }

        private void OnLoadData()
        {
            var backgroundWorker = new BackgroundWorker();
            backgroundWorker.DoWork += (sender, e) =>
                                {
                                    MyData = new List<string> {"Test"};
                                    Thread.Sleep(1000);
                                };
            backgroundWorker.RunWorkerCompleted += (sender, e) => { IsBusy = false; };
            backgroundWorker.RunWorkerAsync();
        }
    }
```

در این ViewModel با اجرای دستور LoadDataCommand داده‌ها از یک منبع داده دریافت می‌شود. این عمل می‌تواند چند ثانیه طول بکشد ، در نتیجه برای قفل نشدن رابط کاربر این عمل را به کمک Background Worker به صورت Async در پشت صحنه انجام شده است.
آزمون واحد این ViewModel اینگونه خواهد بود :

```
[TestFixture]
```

```
public class BackgroundWorkerViewModelTest
{
    #region Setup/Teardown

    [SetUp]
    public void SetUp()
    {
        _backgroundWorkerViewModel = new BackgroundWorkerViewModel();
    }

    #endregion

    private BackgroundWorkerViewModel _backgroundWorkerViewModel;

    [Test]
    public void TestGetData()
    {

        _backgroundWorkerViewModel.LoadDataCommand.Execute(_backgroundWorkerViewModel);

        Assert.NotNull(_backgroundWorkerViewModel.MyData);
        Assert.IsNotEmpty(_backgroundWorkerViewModel.MyData);
    }
}
```

با اجرای این آزمون واحد نتیجه با آن چیزی که در زمان اجرا رخ می‌دهد متفاوت است و با وجود صحیح بودن کدها آزمون واحد شکست می‌خورد.
چون Unit Test به صورت همزمان اجرا می‌شود و برای عملیات‌های پشت صحنه صبر نمی‌کند در نتیجه این آزمون واحد شکست می‌خورد.



یک راه حل تزریق BackgroundWorker به صورت وابستگی به ViewModel می‌باشد. همانطور که قبلا اشاره شده یکی از مزایای استفاده از تکنیک‌های تزریق وابستگی سهولت Unit testing می‌باشد.

در نتیجه یک Interface عمومی و 2 پیاده سازی همزمان و غیر همزمان جهت استفاده در برنامه‌ی واقعی و آزمون واحد تهیه می‌کنیم :

```
public interface IWorker
    {
        void Run(DoWorkEventHandler doWork);
        void Run(DoWorkEventHandler doWork, RunWorkerCompletedEventHandler onComplete);
    }
```

جهت استفاده در برنامه‌ی واقعی :

```
public class AsyncWorker : IWorker
    {
        public void Run(DoWorkEventHandler doWork)
        {
            Run(doWork, null);
        }

        public void Run(DoWorkEventHandler doWork, RunWorkerCompletedEventHandler onComplete)
        {
            var backgroundWorker = new BackgroundWorker();
            backgroundWorker.DoWork += doWork;
            if (onComplete != null)
                backgroundWorker.RunWorkerCompleted += onComplete;
            backgroundWorker.RunWorkerAsync();


        }
    }
```

جهت اجرا در آزمون واحد :

```
public class SyncWorker : IWorker
    {
        #region IWorker Members

        public void Run(DoWorkEventHandler doWork)
        {
            Run(doWork, null);
        }

        public void Run(DoWorkEventHandler doWork, RunWorkerCompletedEventHandler onComplete)
        {
            Exception error = null;
            var doWorkEventArgs = new DoWorkEventArgs(null);
            try
            {
                doWork(this, doWorkEventArgs);
            }
            catch (Exception ex)
            {
                error = ex;
                throw;
            }
            finally
            {
                onComplete(this, new RunWorkerCompletedEventArgs(doWorkEventArgs.Result, error,
doWorkEventArgs.Cancel));
            }
        }

        #endregion
    }
```

در نتیجه ViewModel اینگونه تغییر خواهد کرد :

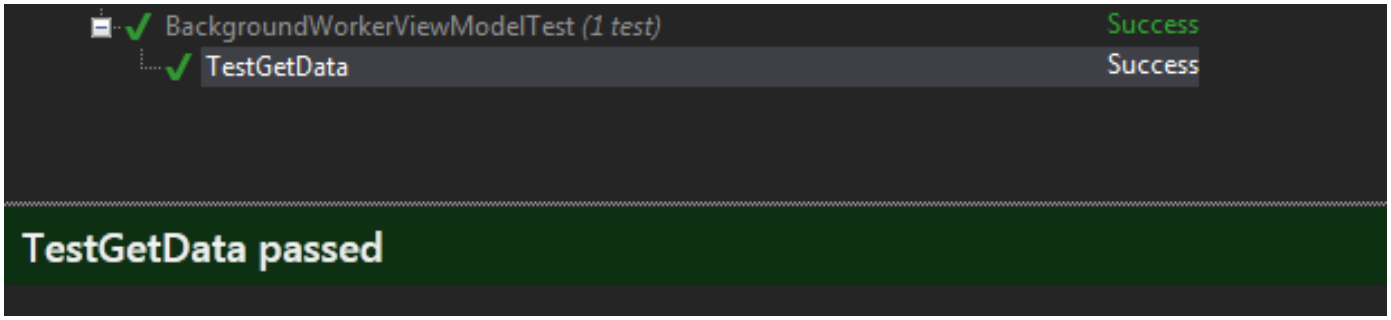
```
public class BackgroundWorkerViewModel : BaseViewModel
    {
        private readonly IWorker _worker;
        private List<string> _myData;
```

```
        public BackgroundWorkerViewModel(IWorker worker)
        {
            _worker = worker;
            LoadDataCommand = new RelayCommand(OnLoadData);
        }

        public RelayCommand LoadDataCommand { get; set; }

        public List<string> MyData
        {
            get { return _myData; }
            set
            {
                _myData = value;
                RaisePropertyChanged(() => MyData);
            }
        }

        public bool IsBusy { get; set; }

        private void OnLoadData()
        {
            IsBusy = true; // view is bound to IsBusy to show 'loading' message.

            _worker.Run(
                (sender, e) =>
                    {
                        MyData = new List<string> {"Test"};
                        Thread.Sleep(1000);
                    },
                (sender, e) => { IsBusy = false; });
        }
    }
```

کلاس مربوطه به آزمون واحد را مطابق با تغییرات ViewModel :

```
[TestFixture]
    public class BackgroundWorkerViewModelTest
    {
        #region Setup/Teardown

        [SetUp]
        public void SetUp()
        {
            _backgroundWorkerViewModel = new BackgroundWorkerViewModel(new SyncWorker());
        }

        #endregion

        private BackgroundWorkerViewModel _backgroundWorkerViewModel;

        [Test]
        public void TestGetData()
        {

            _backgroundWorkerViewModel.LoadDataCommand.Execute(_backgroundWorkerViewModel);

            Assert.NotNull(_backgroundWorkerViewModel.MyData);
            Assert.IsNotEmpty(_backgroundWorkerViewModel.MyData);
        }
    }
```

اکنون اگر Unit Test را اجرا کنیم نتیجه اینگونه خواهد بود :


BackgroundWorkerViewModelTest (1 test)
Success
TestGetData
Success
TestGetData passed