عنوان: آشنایی با mocking frameworks - قسمت دوم
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۲/۱۷ ۱۶:۱۰:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Unit testing

استفاده از mocking frameworks :

تعدادی از چارچوب‌های تقلید نوشته شده برای دات نت فریم ورک مطابق لیست زیر بوده و هدف از آن‌ها ایجاد ساده‌تر اشیاء تقلید برای ما می‌باشد:

Nmock : http://www.nmock.org
Moq : http://code.google.com/p/moq Rhino Mocks : http://ayende.com/projects/rhino-mocks.aspx
TypeMock : http://www.typemock.com
EasyMock.Net : http://sourceforge.net/projects/easymocknet

در این بین Rhino Mocks که توسط یکی از اعضای اصلی تیم NHibernate به وجود آمده است، در مجامع مرتبط بیشتر مورد توجه است. برای آشنایی بیشتر با آن می‌توان به این ویدیوی رایگان آموزشی در مورد آن مراجعه نمود (حدود یک ساعت است).

خلاصه‌ای در مورد نحوه‌ی استفاده از Rhino Mocks :
پس از دریافت کتابخانه سورس باز Rhino Mocks ، ارجاعی را به اسمبلی Rhino.Mocks.dll آن، در پروژه آزمون واحد خود اضافه نمائید.
یک Rhino mock test با ایجاد شیءایی از MockRepository شروع می‌شود و کلا از سه قسمت تشکیل می‌گردد:
الف) ایجاد شیء Mock یا Arrange . هدف از ایجاد شیء mock ، جایگزین کردن و یا تقلید یک شیء واقعی جهت مباحثی مانند ایزوله سازی آزمایشات، بالابردن سرعت آن‌ها و متکی به خود کردن این آزمایشات می‌باشد. همچنین در این حالت نتایج false positive نیز کاهش می‌یابند. منظور از نتایج false positive این است که آزمایش باید با موفقیت به پایان برسد اما اینگونه نشده و علت آن بررسی سیستمی دیگر در خارج از مرزهای سیستم فعلی است و مشکل از جای دیگری نشات گرفته که اساسا هدف از تست ما بررسی عملکرد آن سیستم نبوده است. کلا در این موارد از mocking objects استفاده می‌شود:
- دسترسی به شیء مورد نظر کند است مانند دسترسی به دیتابیس یا محاسبات بسیار طولانی
- شیء مورد نظر از call back استفاده می‌کند

- شیء مورد آزمایش باید به منابع خارجی دسترسی پیدا کند که اکنون مهیا نیستند. برای مثال دسترسی به شبکه.
- شیءایی که می‌خواهیم آن‌را تست کنیم یا برای آن آزمایشات واحد تهیه نمائیم، هنوز کاملا توسعه نیافته و نیمه کاره است.

ب) تعریف رفتارهای مورد نظر یا Act

ج) بررسی رفتارهای تعریف شده یا Assert

مثال:

متد ساده زیر را در نظر بگیرید:

```
public class ImageManagement
  {
      public string GetImageForTimeOfDay()
      {
          int currentHour = DateTime.Now.Hour;
          return currentHour > 6 && currentHour < 21 ? "sun.jpg" : "moon.jpg";
      }

  }
```

آزمایش این متد، وابسته است به زمان جاری سیستم.

```
using System;
using NUnit.Framework;

  [TestFixture]
  public class CMyTest
  {
      [Test]
      public void DaytimeTest()
      {
          int currentHour = DateTime.Now.Hour;

          if (currentHour > 6 && currentHour < 21)
          {
              const string expectedImagePath = "sun.jpg";
              ImageManagement image = new ImageManagement();
              string path = image.GetImageForTimeOfDay();
              Assert.AreEqual(expectedImagePath, path);
          }
          else
          {
              Assert.Ignore("تنها در طول روز قابل بررسی است");
          }
      }

      [Test]
      public void NighttimeTest()
      {
          int currentHour = DateTime.Now.Hour;

          if (currentHour < 6 || currentHour > 21)
          {
              const string expectedImagePath = "moon.jpg";
              ImageManagement image = new ImageManagement();
              string path = image.GetImageForTimeOfDay();
              Assert.AreEqual(expectedImagePath, path);
          }
          else
          {
              Assert.Ignore("تنها در طول شب قابل بررسی است");
          }
      }

  }
```

برای مثال اگر بخواهیم تصویر ماه را دریافت کنیم باید تا ساعت 21 صبر کرد. همچنین بررسی اینکه چرا یکی از متدهای آزمون واحد ما نیز با شکست مواجه شده است نیز نیازمند بررسی زمان جاری است و گاهی ممکن است با شکست مواجه شود و گاهی خیر. در این‌جا با استفاده از یک mock object ، این وضعیت غیرقابل پیش بینی را با منطقی از پیش طراحی شده جایگزین کرده و آزمون خود را بر اساس آن انجام خواهیم داد.

برای این‌کار باید DateTime.Now.Hour را تقلید نموده و اینترفیسی را بر اساس آن طراحی نمائیم. سپس Rhino Mocks کار پیاده سازی این اینترفیس را انجام خواهد داد:

```
using NUnit.Framework;
using Rhino.Mocks;

namespace testWinForms87
{
  public interface IDateTime
  {
      int GetHour();
  }

  public class ImageManagement
  {
      public string GetImageForTimeOfDay(IDateTime time)
      {
          int currentHour = time.GetHour();

          return currentHour > 6 && currentHour < 21 ? "sun.jpg" : "moon.jpg";
      }
  }

  [TestFixture]
  public class CMocking
  {
      [Test]
      public void DaytimeTest()
      {
          MockRepository mocks = new MockRepository();
          IDateTime timeController = mocks.CreateMock<IDateTime>();

          using (mocks.Record())
          {
              Expect.Call(timeController.GetHour()).Return(15);
          }

          using (mocks.Playback())
          {
              const string expectedImagePath = "sun.jpg";
              ImageManagement image = new ImageManagement();
              string path = image.GetImageForTimeOfDay(timeController);
              Assert.AreEqual(expectedImagePath, path);
          }
      }

      [Test]
      public void NighttimeTest()
      {
          MockRepository mocks = new MockRepository();
          IDateTime timeController = mocks.CreateMock<IDateTime>();
          using (mocks.Record())
          {
              Expect.Call(timeController.GetHour()).Return(1);
          }

          using (mocks.Playback())
          {
              const string expectedImagePath = "moon.jpg";
              ImageManagement image = new ImageManagement();
              string path = image.GetImageForTimeOfDay(timeController);
              Assert.AreEqual(expectedImagePath, path);
          }
      }
  }

}
```

همانطور که در ابتدای مطلب هم عنوان شد، mocking از سه قسمت تشکیل می‌شود:

```
MockRepository mocks = new MockRepository();
```

ابتدا شیء mocks را از MockRepository کتابخانه Rhino Mocks ایجاد می‌کنیم تا بتوان از خواص و متدهای آن استفاده کرد.
سپس اینترفیسی باید به آن پاس شود تا انتظارات سیستم را بتوان در آن بر پا نمود:

```
IDateTime timeController = mocks.CreateMock<IDateTime>();
using (mocks.Record())
{
 Expect.Call(timeController.GetHour()).Return(15);

}
```

به عبارت دیگر در اینجا به سیستم مقلد خود خواهیم گفت: زمانیکه شیء ساعت را تقلید کردی، لطفا عدد 15 را برگردان.
به این صورت آزمایش ما بر اساس وضعیت مشخصی از سیستم صورت می‌گیرد و وابسته به ساعت جاری سیستم نخواهد بود.

همانطور که ملاحظه می‌کنید، روش Test Driven Development بر روی نحوه‌ی برنامه نویسی ما و ایجاد کلاس‌ها و اینترفیس‌های اولیه نیز تاثیر زیادی خواهد گذاشت. استفاده از اینترفیس‌ها یکی از اصول پایه‌ای برنامه نویسی شیءگرا است و در اینجا مقید به ایجاد آن‌ها خواهیم شد.

پس از آن‌که در قسمت mocks.Record ، انتظارات خود را ثبت کردیم، اکنون نوبت به وضعیت Playback می‌رسد:

```
using (mocks.Playback())
{
 string expectedImagePath = "sun.jpg";
 ImageManagement image = new ImageManagement();
 string path = image.GetImageForTimeOfDay(timeController);
 Assert.AreEqual(expectedImagePath, path);

}
```

در اینجا روش کار همانند ایجاد متدهای آزمون واحد متداولی است که تاکنون با آن‌ها آشنا شده‌ایم و تفاوتی ندارد.
با توجه به اینکه پس از تغییر طراحی متد GetImageForTimeOfDay ، این متد اکنون از شیء IDateTime به عنوان ورودی استفاده می‌کند، می‌توان پیاده سازی آن اینترفیس را در آزمایشات واحد تقلید نمود و یا جایی که قرار است در برنامه استفاده شود، می‌تواند پیاده سازی واقعی خود را داشته باشد و دیگر آزمایشات ما وابسته به آن نخواهد بود:

```
public class DateTimeController : IDateTime
{
  public int GetHour()
  {
     return DateTime.Now.Hour;
  }
}
```

نظرات خوانندگان

نویسنده: SirAsad
تاریخ: ۱۳۸۸/۰۲/۱۹ ۱۵:۰۰:۰۰

آقا بسیار مطلب جالبی بود... در این مورد در بلاگ یه مطلب گذاشتم ... امیدوارم مفید واقع بشه.

http://sir.blogsky.com/1388/02/19/post-115

موفق باشید