عنوان: **Roslyn #7**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۷/۰۱ ۱۵:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Roslyn

معرفی Workspace API

Workspace، در حقیقت نمایش اجزای یک Solution در ویژوال استودیو است و یک Solution متشکل است از تعدادی پروژه به همراه وابستگی‌های بین آن‌ها. هدف از وجود Workspace API در Roslyn، دسترسی به اطلاعات لازم جهت انجام امور Refactoring در سطح یک Solution است. برای مثال اگر قرار است نام خاصیتی تغییر کند و این خاصیت در چندین پروژه‌ی دیگر در حال استفاده است، این نام باید در سراسر Solution جاری یافت شده و تغییر یابد. همچنین برفراز Workspace API تعدادی سرویس زبان مانند فرمت کننده‌های کدها، تغییرنام دهنده‌های سیمبل‌ها و توصیه کننده‌ها نیز تهیه شده‌اند.
همچنین این سرویس‌ها و API تهیه شده، منحصر به ویژوال استودیو نیستند و VS 2015 تنها از آن‌ها استفاده می‌کند. برای مثال نگارش‌های جدیدتر mono-develop لینوکسی نیز شروع به استفاده‌ی از Roslyn کرده‌اند.

نمایش اجزای یک Solution

در ادامه مثالی را مشاهده می‌کنید که توسط آن نام Solution و سپس تمام پروژه‌های موجود در آن‌ها به همراه نام فایل‌های مرتبط و همچنین ارجاعات آن‌ها در صفحه نمایش داده می‌شوند:

```
var ws = MSBuildWorkspace.Create();
var sln = ws.OpenSolutionAsync(@"..\..\..\Roslyn.sln").Result;


// Print the root of the solution.
Console.WriteLine(Path.GetFileName(sln.FilePath));


// Get dependency graph to perform a sort.
var g = sln.GetProjectDependencyGraph();
var ps = g.GetTopologicallySortedProjects();


// Print all projects, their documents, and references.
foreach (var p in ps)
{
    var proj = sln.GetProject(p);

    Console.WriteLine("> " + proj.Name);

    Console.WriteLine("  > References");
    foreach (var r in proj.ProjectReferences)
    {
        Console.WriteLine("    - " + sln.GetProject(r.ProjectId).Name);
    }

    foreach (var d in proj.Documents)
    {
        Console.WriteLine("  - " + d.Name);
    }
}
```

در ابتدا نیاز است یک وهله از MSBuildWorkspace را ایجاد کرد. اکنون با استفاده از این Workspace می‌توان solution خاصی را گشود و آنالیز کرد. قسمتی از خروجی آن چنین شکلی را دارد:

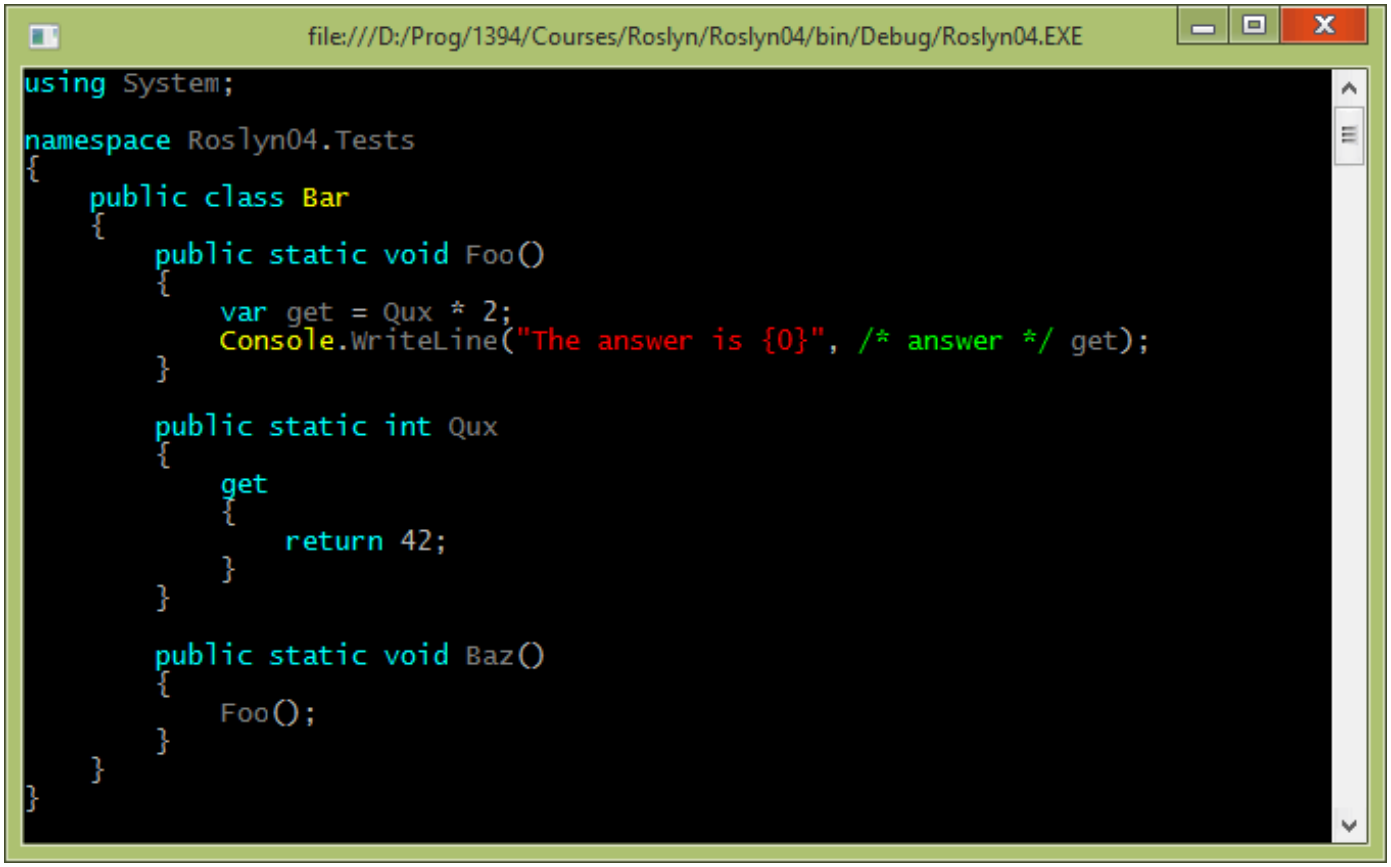
```
Roslyn.sln
> Roslyn01
  > References
  - Program.cs
  - AssemblyInfo.cs
  - .NETFramework,Version=v4.6.AssemblyAttributes.cs
```

ایجاد یک Syntax highlighter با استفاده از Classification service

هدف از Classification service، رندر کردن فایل‌ها در ادیتور جاری است. برای این منظور نیاز است بتوان واژه‌های کلیدی، کامنت‌ها، نام‌های نوع‌ها و امثال آن‌ها را به صورت کلاسه شده در اختیار داشت و سپس برای مثال هرکدام را با رنگی مجزا نمایش داد و رندر کرد.
در ادامه مثالی از آن‌را ملاحظه می‌کنید:

```
var ws = MSBuildWorkspace.Create();
var sln = ws.OpenSolutionAsync(@"..\..\..\Roslyn.sln").Result;

// Get the Tests\Bar.cs document.
var proj = sln.Projects.Single(p => p.Name == "Roslyn04.Tests");
var test = proj.Documents.Single(d => d.Name == "Bar.cs");

var tree = test.GetSyntaxTreeAsync().Result;
var root = tree.GetRootAsync().Result;

// Get all the spans in the document that are classified as language elements.
var spans = Classifier.GetClassifiedSpansAsync(test, root.FullSpan).Result.ToDictionary(c =>
c.TextSpan.Start, c => c);

// Print the source text with appropriate colorization.
var txt = tree.GetText().ToString();

var i = 0;
foreach (var c in txt)
{
    var span = default(ClassifiedSpan);
    if (spans.TryGetValue(i, out span))
    {
        var color = ConsoleColor.Gray;

        switch (span.ClassificationType)
        {
            case ClassificationTypeNames.Keyword:
                color = ConsoleColor.Cyan;
                break;
            case ClassificationTypeNames.StringLiteral:
            case ClassificationTypeNames.VerbatimStringLiteral:
                color = ConsoleColor.Red;
                break;
            case ClassificationTypeNames.Comment:
                color = ConsoleColor.Green;
                break;
            case ClassificationTypeNames.ClassName:
            case ClassificationTypeNames.InterfaceName:
            case ClassificationTypeNames.StructName:
            case ClassificationTypeNames.EnumName:
            case ClassificationTypeNames.TypeParameterName:
            case ClassificationTypeNames.DelegateName:
                color = ConsoleColor.Yellow;
                break;
            case ClassificationTypeNames.Identifier:
                color = ConsoleColor.DarkGray;
                break;
        }

        Console.ForegroundColor = color;
    }

    Console.Write(c);

    i++;
}
```

با این خروجی:



توضیحات:

در اینجا نیز کار با ایجاد یک Workspace و سپس گشودن Solution ایی مشخص در آن آغاز می‌شود. سپس در آن به دنبال پروژه‌ای به نام Roslyn04.Tests می‌گردیم. این پروژه حاوی تعدادی کلاس، جهت بررسی و آزمایش هستند. برای مثال در اینجا فایل Bar.cs آن قرار است آنالیز شود. پس از یافتن آن، ابتدا syntax tree آن دریافت می‌گردد و سپس به سرویس Classifier.GetClassifiedSpansAsync ارسال خواهد شد. خروجی آن شامل لیستی از Classified Spans است؛ مانند کلمات کلیدی، رشته‌ها، کامنت‌ها و غیره. در ادامه این لیست تبدیل به یک دیکشنری می‌شود که کلید آن محل آغاز این span و مقدار آن، مقدار span است. سپس متن syntax tree دریافت شده و حرف به حرف آن در طی یک حلقه بررسی می‌شود. در این حلقه، مقدار i به محل حروف جاری مورد آنالیز اشاره می‌کند. اگر این محل در دیکشنری Classified Spans وجود داشت، یعنی یک span جدید شروع شده‌است و بر این اساس، نوع آن span را می‌توان استخراج کرد و سپس بر اساس این نوع، رنگ متفاوتی را در صفحه نمایش داد.

سرویس فرمت کردن کدها

این سرویس کار فرمت خودکار کدهای بهم ریخته را انجام می‌دهد؛ مانند تنظیم فاصله‌های خالی و یا ایجاد indentation و امثال آن. در حقیقت Ctlr K+D در ویژوال استودیو، دقیقا از همین سرویس زبان استفاده می‌کند.

کار کردن با این سرویس از طریق برنامه نویسی به نحو ذیل است:

```
var ws = MSBuildWorkspace.Create();
var sln = ws.OpenSolutionAsync(@"..\..\..\Roslyn.sln").Result;


// Get the Tests\Qux.cs document.
var proj = sln.Projects.Single(p => p.Name == "Roslyn04.Tests");
var qux = proj.Documents.Single(d => d.Name == "Qux.cs");

Console.WriteLine("Before:");
Console.WriteLine();
Console.WriteLine(qux.GetSyntaxTreeAsync().Result.GetText());
```

```
Console.WriteLine();
Console.WriteLine();


// Apply formatting and print the result.
var res = Formatter.FormatAsync(qux).Result;

Console.WriteLine("After:");
Console.WriteLine();
Console.WriteLine(res.GetSyntaxTreeAsync().Result.GetText());
Console.WriteLine();
```

با این خروجی:

```
Before:

using System;

namespace Roslyn04.Tests
{
    class Qux {
        public void Baz()
        { Console.WriteLine(42);
            return;  }
    }
}


After:

using System;

namespace Roslyn04.Tests
{
    class Qux
    {
        public void Baz()
        {
            Console.WriteLine(42);
            return;
        }
    }
}
```

همانطور که ملاحظه می‌کنید، فایل Qux.cs که فرمت مناسبی ندارد. بنابراین باز شده و syntax tree آن به سرویس Formatter.FormatAsync جهت فرمت شدن ارسال می‌شود.

سرویس یافتن سیمبل‌ها

یکی دیگر از قابلیت‌هایی که در ویژوال استودیو وجود دارد، امکان یافتن سیمبل‌ها است. برای مثال این نوع یا کلاس خاص، در کجاها استفاده شده‌است و به آن ارجاعاتی وجود دارد. مواردی مانند Find all references، Go to definition و نمایش Call hierarchy از این سرویس استفاده می‌کنند.

```
var ws = MSBuildWorkspace.Create();
var sln = ws.OpenSolutionAsync(@"..\..\..\Roslyn.sln").Result;


// Get the Tests project.
var proj = sln.Projects.Single(p => p.Name == "Roslyn04.Tests");

// Locate the symbol for the Bar.Foo method and the Bar.Qux property.
var comp = proj.GetCompilationAsync().Result;

var barType = comp.GetTypeByMetadataName("Roslyn04.Tests.Bar");

var fooMethod = barType.GetMembers().Single(m => m.Name == "Foo");
var quxProp = barType.GetMembers().Single(m => m.Name == "Qux");
```

```
// Find callers across the solution.
Console.WriteLine("Find callers of Foo");
Console.WriteLine();

var callers = SymbolFinder.FindCallersAsync(fooMethod, sln).Result;
foreach (var caller in callers)
{
    Console.WriteLine(caller.CallingSymbol);
    foreach (var location in caller.Locations)
    {
        Console.WriteLine("    " + location);
    }
}

Console.WriteLine();
Console.WriteLine();

// Find all references across the solution.
Console.WriteLine("Find all references to Qux");
Console.WriteLine();

var references = SymbolFinder.FindReferencesAsync(quxProp, sln).Result;
foreach (var reference in references)
{
    Console.WriteLine(reference.Definition);
    foreach (var location in reference.Locations)
    {
        Console.WriteLine("    " + location.Location);
    }
}
```

در این مثال، پروژه‌ی Roslyn04.Tests که حاوی کلاس‌های Foo و Qux است، جهت آنالیز باز شده‌است. در اینجا برای رسیدن به Symbols نیاز است ابتدا به Compilation API دسترسی یافت و سپس متادیتاها را بر اساس آن استخراج کرد. سپس متدهای Foo و خاصیت Qux آن یافت شده‌اند.

اکنون با استفاده از سرویس SymbolFinder.FindCallersAsync تمام فراخوان‌های متد Foo را در سراسر Solution جاری می‌یابیم.

سپس با استفاده از سرویس SymbolFinder.FindReferencesAsync تمام ارجاعات به خاصیت Qux را در Solution جاری نمایش می‌دهیم.

سرویس توصیه کننده

Intellisense در ویژوال استودیو از سرویس توصیه کننده‌ی Roslyn استفاده می‌کند.

```
var ws = MSBuildWorkspace.Create();
var sln = ws.OpenSolutionAsync(@"..\..\..\Roslyn.sln").Result;

// Get the Tests\Foo.cs document.
var proj = sln.Projects.Single(p => p.Name == "Roslyn04.Tests");
var foo = proj.Documents.Single(d => d.Name == "Foo.cs");


// Find the 'dot' token in the first Console.WriteLine member access expression.
var tree = foo.GetSyntaxTreeAsync().Result;
var model = proj.GetCompilationAsync().Result.GetSemanticModel(tree);
var consoleDot =
tree.GetRoot().DescendantNodes().OfType<MemberAccessExpressionSyntax>().First().OperatorToken;


// Get recommendations at the indicated cursor position.
//
//   Console.WriteLine
//           ^
var res = Recommender.GetRecommendedSymbolsAtPosition(

                    model, consoleDot.GetLocation().SourceSpan.Start + 1, ws).ToList();

foreach (var rec in res)
{
    Console.WriteLine(rec);
}
```

در این مثال سعی شده‌است لیست توصیه‌های ارائه شده در حین تایپ دات، توسط سرویس Recommender.GetRecommendedSymbolsAtPosition دریافت و نمایش داده شوند. در ابتدای کار، کلاس Foo گشوده شده و سپس Syntax tree و Semantic model آن استخراج می‌شود. این model پارامتر اول متد سرویس توصیه کننده‌است. سپس نیاز است محل مکانی را به آن معرفی کنیم تا کار توصیه کردن را بر اساس آن شروع کند. برای نمونه در اینجا OperatorToken در حقیقت همان دات مربوط به Console.WriteLine است. پس از یافتن این توکن، امکان دسترسی به مکان آن وجود دارد.
تعدادی از خروجی‌های مثال فوق به صورت زیر هستند:

```
System.Console.Beep()
System.Console.Beep(int, int)
System.Console.Clear()
```

سرویس تغییر نام دادن

هدف از سرویس Renamer.RenameSymbolAsync، تغییر نام یک identifier در کل Solution است. نمونه‌ای از نحوه‌ی کاربرد آن‌را در مثال ذیل مشاهده می‌کنید:

```
var ws = MSBuildWorkspace.Create();
var sln = ws.OpenSolutionAsync(@"..\..\..\Roslyn.sln").Result;


// Get Tests\Bar.cs before making changes.
var oldProj = sln.Projects.Single(p => p.Name == "Roslyn04.Tests");
var oldDoc = oldProj.Documents.Single(d => d.Name == "Bar.cs");

Console.WriteLine("Before:");
Console.WriteLine();

var oldTxt = oldDoc.GetTextAsync().Result;
Console.WriteLine(oldTxt);

Console.WriteLine();
Console.WriteLine();


// Get the symbol for the Bar.Foo method.
var comp = oldProj.GetCompilationAsync().Result;

var barType = comp.GetTypeByMetadataName("Roslyn04.Tests.Bar");
var fooMethod = barType.GetMembers().Single(m => m.Name == "Foo");


// Perform the rename.
var newSln = Renamer.RenameSymbolAsync(sln, fooMethod, "Foo2", ws.Options).Result;


// Get Tests\Bar.cs after making changes.
var newProj = newSln.Projects.Single(p => p.Name == "Roslyn04.Tests");
var newDoc = newProj.Documents.Single(d => d.Name == "Bar.cs");

Console.WriteLine("After:");
Console.WriteLine();

var newTxt = newDoc.GetTextAsync().Result;
Console.WriteLine(newTxt);
```

در این مثال، متد Foo کلاس Bar، قرار است به Foo2 تغییرنام یابد. به همین منظور ابتدا پروژه‌ی حاوی فایل Bar.cs باز شده و اطلاعات این کلاس استخراج می‌گردد. سپس اصل این کلاس تغییر نیافته نمایش داده می‌شود. در ادامه با استفاده از API کامپایل، به متادیتای متد Foo یا به عبارتی Symbol آن دسترسی پیدا می‌کنیم. سپس این Symbol به متد یا سرویس Renamer.RenameSymbolAsync ارسال می‌شود تا کار تغییر نام صورت گیرد. پس از اینکار مجددا متن کلاس تغییر یافته نمایش داده خواهد شد.

سرویس ساده کننده

هدف از سرویس ساده کننده، ساده‌کردن و کاهش کدهای ارائه شده، از دید Semantics است. برای مثال اگر فضای نامی در قسمت using ذکر شده‌است، دیگر نیازی نیست تا این فضای نام به ابتدای فراخوانی یک متد آن اضافه شود و می‌توان این قطعه از کد را ساده‌تر کرد و کاهش داد.

```
var ws = MSBuildWorkspace.Create();
var sln = ws.OpenSolutionAsync(@"..\..\..\Roslyn.sln").Result;


// Get the Tests\Baz.cs document.
var proj = sln.Projects.Single(p => p.Name == "Roslyn04.Tests");
var baz = proj.Documents.Single(d => d.Name == "Baz.cs");

Console.WriteLine("Before:");
Console.WriteLine();
Console.WriteLine(baz.GetSyntaxTreeAsync().Result.GetText());

Console.WriteLine();
Console.WriteLine();

var oldRoot = baz.GetSyntaxRootAsync().Result;


var memberAccesses = oldRoot.DescendantNodes().OfType<CastExpressionSyntax>();
var newRoot = oldRoot.ReplaceNodes(memberAccesses, (_, m) =>
m.WithAdditionalAnnotations(Simplifier.Annotation));

var newDoc = baz.WithSyntaxRoot(newRoot);


// Invoke the simplifier and print the result.
var res = Simplifier.ReduceAsync(newDoc).Result;

Console.WriteLine("After:");
Console.WriteLine();
Console.WriteLine(res.GetSyntaxTreeAsync().Result.GetText());
Console.WriteLine();
```

در این مثال نحوه‌ی ساده سازی castهای اضافی را ملاحظه می‌کنید. برای مثال اگر نوع متغیری int است، دیگر نیازی نیست در سراسر کد در کنار این متغیر، cast به int را هم ذکر کرد و می‌توان این کد را ساده‌تر نمود.

کدهای کامل این سری را از اینجا می‌توانید دریافت کنید:

Roslyn-Samples.zip