EBook Gratis

APRENDIZAJE roslyn

Free unaffiliated eBook created from
Stack Overflow contributors.

#roslyn

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: roslyn

It is an unofficial and free roslyn ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official roslyn.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Roslyn

Observaciones

Para comenzar con Roslyn, eche un vistazo a:

- API del árbol de sintaxis
- Modelo semántico API
- *Añadir más temas aquí* .

Examples

Instalación o configuración

Para comenzar a jugar con Roslyn, necesitará los siguientes paquetes de NuGet:

- Los compiladores de C # y VB - `Microsoft.Net.Compilers` . Para instalarlo, puede ejecutar el siguiente comando en la Consola del Administrador de paquetes:

  ```
  nuget install Microsoft.Net.Compilers
  ```

- Las API y servicios de lenguaje: `Microsoft.CodeAnalysis` . Para instalarlo, puede ejecutar el siguiente comando en la Consola del Administrador de paquetes:

  ```
  nuget install Microsoft.CodeAnalysis
  ```

Además, es bueno instalar las Plantillas del SDK de la plataforma del compilador .NET, que se pueden encontrar aquí . Esto te llevará a:

- Plantillas para C # y Visual Basic que permiten la creación de analizadores, CodeFixes y herramientas de análisis independientes.
- La herramienta Visualizador de sintaxis para Visual Studio (`View -> Other Windows -> Syntax Visualizer`), que es extremadamente útil para examinar el árbol de sintaxis del código existente.

Herramientas y recursos adicionales

- El ques de roslyn

Una herramienta para convertir un programa de C # de muestra en sintaxis de llamadas a API. La herramienta en sí se puede encontrar aquí .

- Visor fuente mejorado

Una forma fácil de ver el código fuente de Roslyn se puede encontrar aquí .

Lea Empezando con Roslyn en línea: https://riptutorial.com/es/roslyn/topic/2905/empezando-con-

roslyn

Capítulo 2: Analizar el código fuente con Roslyn

Examples

Análisis introspectivo de un analizador en C

1. Crear una nueva **aplicación de consola**
2. Agregue el paquete **NuGet** `Microsoft.CodeAnalysis`
3. Importe los espacios de nombres `Microsoft.CodeAnalysis.MSBuild` , `System.Linq` y `Microsoft.CodeAnalysis.CSharp.Syntax`
4. Escriba el siguiente código de ejemplo en el método `Main` :

```
// Declaring a variable with the current project file path.
const string projectPath = @"C:\<your path to the project\<project file name>.csproj";

// Creating a build workspace.
var workspace = MSBuildWorkspace.Create();

// Opening this project.
var project = workspace.OpenProjectAsync(projectPath).Result;

// Getting the compilation.
var compilation = project.GetCompilationAsync().Result;

// As this is a simple single file program, the first syntax tree will be the current file.
var syntaxTree = compilation.SyntaxTrees.First();

// Getting the root node of the file.
var rootSyntaxNode = syntaxTree.GetRootAsync().Result;

// Finding all the local variable declarations in this file and picking the first one.
var firstLocalVariablesDeclaration =
rootSyntaxNode.DescendantNodesAndSelf().OfType<LocalDeclarationStatementSyntax>().First();

// Getting the first declared variable in the declaration syntax.
var firstVariable = firstLocalVariablesDeclaration.Declaration.Variables.First();

// Getting the text of the initialized value.
var variableInitializer = firstVariable.Initializer.Value.GetFirstToken().ValueText;

// This will print to screen the value assigned to the projectPath variable.
Console.WriteLine(variableInitializer);

Console.ReadKey();
```

Al ejecutar el proyecto, verá la variable declarada en la parte superior impresa en la pantalla. Esto significa que usted analizó un proyecto con éxito y encontró una variable en él.

Analizar una aplicación simple "Hello World" en C

Cree una nueva aplicación de consola con una línea en el método `Main` : `Console.WriteLine("Hello`

`World")`

Recuerde la ruta al archivo `.csproj` y reemplácelo en el ejemplo.

Cree una nueva **aplicación de consola** e instale el paquete `Microsoft.CodeAnalysis` NuGet y pruebe el siguiente código:

```
const string projectPath = @"C:\HelloWorldApplication\HelloWorldProject.csproj";

// Creating a build workspace.
var workspace = MSBuildWorkspace.Create();

// Opening the Hello World project.
var project = workspace.OpenProjectAsync(projectPath).Result;

// Getting the compilation.
var compilation = project.GetCompilationAsync().Result;

foreach (var tree in compilation.SyntaxTrees)
{
    Console.WriteLine(tree.FilePath);

    var rootSyntaxNode = tree.GetRootAsync().Result;

    foreach (var node in rootSyntaxNode.DescendantNodes())
    {
        Console.WriteLine($" *** {node.Kind()}");
        Console.WriteLine($"     {node}");
    }
}

Console.ReadKey();
```

Esto imprimirá todos los archivos y todos los **nodos de sintaxis** en su proyecto **Hello World** .

Analizar una aplicación simple "Hello World" en VB.NET

Cree una nueva aplicación de consola con una línea en el método `Main` : `Console.WriteLine("Hello World")`

Recuerde la ruta al archivo `.vbproj` y reemplácelo en el ejemplo.

Cree una nueva **aplicación de consola** e instale el paquete `Microsoft.CodeAnalysis` NuGet y pruebe el siguiente código:

```
Const projectPath = "C:\HelloWorldApplication\HelloWorldProject.vbproj"

' Creating a build workspace.
Dim workspace = MSBuildWorkspace.Create()

' Opening the Hello World project.
Dim project = workspace.OpenProjectAsync(projectPath).Result

' Getting the compilation.
Dim compilation = project.GetCompilationAsync().Result

For Each tree In compilation.SyntaxTrees
```

```
    Console.WriteLine(tree.FilePath)

    Dim rootSyntaxNode = tree.GetRootAsync().Result

    For Each node In rootSyntaxNode.DescendantNodes()

        Console.WriteLine($" *** {node.Kind()}")
        Console.WriteLine($"     {node}")
    Next
Next

Console.ReadKey()
```

Esto imprimirá todos los archivos y todos los **nodos de sintaxis** en su proyecto **Hello World** .

Analizar el código fuente del texto en C

```
var syntaxTree = CSharpSyntaxTree.ParseText(
@"using System;
using System.Collections;
using System.Linq;
using System.Text;

namespace HelloWorldApplication
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine(""Hello World"");
}
}
}");

var root = syntaxTree.GetRoot() as CompilationUnitSyntax;

var namespaceSyntax = root.Members.OfType<NamespaceDeclarationSyntax>().First();

var programClassSyntax = namespaceSyntax.Members.OfType<ClassDeclarationSyntax>().First();

var mainMethodSyntax = programClassSyntax.Members.OfType<MethodDeclarationSyntax>().First();

Console.WriteLine(mainMethodSyntax.ToString());

Console.ReadKey();
```

Este ejemplo imprimirá el método `Main` del texto que analiza la sintaxis.

Obtener el tipo de 'var'

Para obtener el tipo real para una variable declarada usando `var` , llame a `GetSymbolInfo()` en el `SemanticModel` . Puede abrir una solución existente usando `MSBuildWorkspace` , luego enumerar sus proyectos y sus documentos. Use un documento para obtener su `SyntaxRoot` y `SemanticModel` , luego busque `VariableDeclarations` y recupere los símbolos para el `Type` de una variable declarada como esta:

```
var workspace = MSBuildWorkspace.Create();
var solution = workspace.OpenSolutionAsync("c:\\path\\to\\solution.sln").Result;

foreach (var document in solution.Projects.SelectMany(project => project.Documents))
{
    var rootNode = document.GetSyntaxRootAsync().Result;
    var semanticModel = document.GetSemanticModelAsync().Result;

    var variableDeclarations = rootNode
            .DescendantNodes()
            .OfType<LocalDeclarationStatementSyntax>();
    foreach (var varDeclaration in variableDeclarations)
    {
        var symbolInfo = semanticModel.GetSymbolInfo(varDeclaration.Declaration.Type);
        var typeSymbol = symbolInfo.Symbol; // the type symbol for the variable..
    }
}
```

Lea Analizar el código fuente con Roslyn en línea:
https://riptutorial.com/es/roslyn/topic/4712/analizar-el-codigo-fuente-con-roslyn

Capítulo 3: Árbol de sintaxis

Introducción

Una de las partes principales del compilador de Roslyn es la API de sintaxis. Expone los árboles de sintaxis que los compiladores utilizan para comprender los programas de Visual Basic y C #.

Observaciones

- El árbol de sintaxis es un árbol de análisis en el contexto del compilador de Roslyn.

Examples

Obtención de la raíz del árbol de sintaxis de un documento.

Si ya tiene acceso a su clase de `Document` desde su área de trabajo (usando espacios de trabajo), es fácil acceder a la raíz de su árbol de sintaxis.

```
Document document = ... // Get document from workspace or other source

var syntaxRoot = await document.GetSyntaxRootAsync();
```

Atravesando el árbol de sintaxis usando LINQ

Puede navegar fácilmente por el árbol de sintaxis utilizando LINQ. Por ejemplo, es fácil obtener todos los nodos `ClassDeclarationSyntax` (clases declaradas), que tienen un nombre que comienza con la letra `A` :

```
var allClassesWithNameStartingWithA = syntaxRoot.DescendantNodes()
    .OfType<ClassDeclarationSyntax>()
    .Where(x => x.Identifier.ToString().StartsWith("A"));
```

O obteniendo todas las clases que tienen atributos:

```
var allClassesWithAttriutes = syntaxRoot.DescendantNodes()
    .OfType<ClassDeclarationSyntax>()
    .Where(x => x.AttributeLists.Any(y => y.Attributes.Any()));
```

Atravesando el árbol de sintaxis usando un CSharpSyntaxWalker

La clase `CSharpSyntaxWalker` está fuera de la implementación del cuadro de patrón de visitante, que podemos usar para atravesar nuestro árbol de sintaxis. Aquí hay un ejemplo simple de un Syntax Walker que recopila todas las `struct` que tienen un nombre, comenzando con la letra `A` :

```
public class StructCollector : CSharpSyntaxWalker
```

```
{
    public StructCollector()
    {
        this.Structs = new List<StructDeclarationSyntax>();
    }

    public IList<StructDeclarationSyntax> Structs { get; }

    public override void VisitStructDeclaration(StructDeclarationSyntax node)
    {
        if (node.Identifier.ToString().StartsWith("A"))
        {
            this.Structs.Add(node);
        }
    }
}
```

Podemos usar nuestro SyntaxWalker de la siguiente manera:

```
var structCollector = new StructCollector();
structCollector.Visit(syntaxRoot); // Or any other syntax node
Console.WriteLine($"The number of structs that have a name starting with the letter 'A' is
{structCollector.Structs.Count}");
```

Lea Árbol de sintaxis en línea: https://riptutorial.com/es/roslyn/topic/9765/arbol-de-sintaxis

Capítulo 4: Cambiar el código fuente con Roslyn

Introducción

Ejemplos prácticos de usar Roslyn para transformaciones de código fuente.

Observaciones

- Los árboles de sintaxis de Roslyn son inmutables. Al llamar a un método como ReplaceNodes, generamos un nuevo nodo en lugar de modificar el existente. Esto requiere que siempre cambies el objeto en el que has estado trabajando.

Examples

Reemplace los atributos existentes para todos los métodos en C # usando el árbol de sintaxis

El siguiente fragmento de código reemplaza todos los Atributos llamados `PreviousAttribute` por un Atributo llamado `ReplacementAttribute` para una solución completa. La muestra busca manualmente el árbol de Sintaxis y reemplaza todos los nodos afectados.

```
    static async Task<bool> ModifySolution(string solutionPath)
    {
        using (var workspace = MSBuildWorkspace.Create())
        {
            // Selects a Solution File
            var solution = await workspace.OpenSolutionAsync(solutionPath);
            // Iterates through every project
            foreach (var project in solution.Projects)
            {
                // Iterates through every file
                foreach (var document in project.Documents)
                {
                    // Selects the syntax tree
                    var syntaxTree = await document.GetSyntaxTreeAsync();
                    var root = syntaxTree.GetRoot();
                    // Finds all Attribute Declarations in the Document
                    var existingAttributesList =
root.DescendantNodes().OfType<AttributeListSyntax>()
                        // Where the Attribute is declared on a method
                        .Where(curr => curr.Parent is MethodDeclarationSyntax)
                        // And the attribute is named "PreviousAttribute"
                        .Where(curr => curr.Attributes.Any(currentAttribute =>
currentAttribute.Name.GetText().ToString() == "PreviousAttribute"))
                        .ToList();
                    if (existingAttributesList.Any())
                    {
                        // Generates a replacement for every attribute
```

```
                        var replacementAttribute = SyntaxFactory.AttributeList(
                            SyntaxFactory.SingletonSeparatedList(

SyntaxFactory.Attribute(SyntaxFactory.IdentifierName("ReplacementAttribute"),
                                    SyntaxFactory.AttributeArgumentList(
                                        SyntaxFactory.SeparatedList(new[]
                                        {
                                        SyntaxFactory.AttributeArgument(
                                            SyntaxFactory.LiteralExpression(
                                                SyntaxKind.StringLiteralExpression,
SyntaxFactory.Literal(@"Sample"))
                                            )
                                        }))))));
                        // Replaces all attributes at once.
                        // Note that you should not use root.ReplaceNode
                        // since it would only replace the first note
                        root = root.ReplaceNodes(existingAttributesList, (node, n2) =>
replacementAttribute);
                        // Exchanges the document in the solution by the newly generated
document
                        solution = solution.WithDocumentSyntaxRoot(document.Id, root);
                    }
                }
            }
            // applies the changes to the solution
            var result = workspace.TryApplyChanges(solution);
            return result;
        }
    }
```

El ejemplo anterior se puede probar para la siguiente clase:

```
public class Program
{
    [PreviousAttribute()]
    static void Main(string[] args)
    {
    }
}
```

No debe utilizar Methode root.ReplaceNode para reemplazar varios nodos. Como el árbol es inmutable, estarás trabajando en diferentes objetos. El uso del siguiente fragmento de código en el ejemplo anterior no generaría el resultado esperado:

```
foreach(var node in existingAttributesList){
    root = root.ReplaceNode(node, replacementAttribute);
}
```

La primera llamada a `ReplaceNode` crearía un nuevo elemento raíz. Sin embargo, los elementos en `existingAttributesList` pertenecen a una raíz diferente (el elemento raíz anterior) y no pueden reemplazarse debido a esto. Esto daría como resultado que el primer atributo se reemplace y los siguientes atributos permanezcan sin cambios, ya que todas las llamadas consecutivas se realizarán en un nodo que no esté presente en el nuevo árbol.

Reemplace los atributos existentes para todos los métodos en C # usando un

SyntaxRewriter

El siguiente fragmento de código reemplaza todos los Atributos llamados "Atributo Anterior" por un Atributo llamado "Atributo de Reemplazo" para una solución completa. La muestra utiliza manualmente un SyntaxRewriter para intercambiar los atributos.

```
/// <summary>
/// The CSharpSyntaxRewriter allows to rewrite the Syntax of a node
/// </summary>
public class AttributeStatementChanger : CSharpSyntaxRewriter
{
    /// Visited for all AttributeListSyntax nodes
    /// The method replaces all PreviousAttribute attributes annotating a method by
ReplacementAttribute attributes
    public override SyntaxNode VisitAttributeList(AttributeListSyntax node)
    {
        // If the parent is a MethodDeclaration (= the attribute annotes a method)
        if (node.Parent is MethodDeclarationSyntax &&
            // and if the attribute name is PreviousAttribute
            node.Attributes.Any(
                currentAttribute => currentAttribute.Name.GetText().ToString() ==
"PreviousAttribute"))
        {
            // Return an alternate node that is injected instead of the current node
            return SyntaxFactory.AttributeList(
                            SyntaxFactory.SingletonSeparatedList(

SyntaxFactory.Attribute(SyntaxFactory.IdentifierName("ReplacementAttribute"),
                                SyntaxFactory.AttributeArgumentList(
                                    SyntaxFactory.SeparatedList(new[]
                                    {
                                    SyntaxFactory.AttributeArgument(
                                        SyntaxFactory.LiteralExpression(
                                            SyntaxKind.StringLiteralExpression,
SyntaxFactory.Literal(@"Sample"))
                                        )
                                    }))))));
        }
        // Otherwise the node is left untouched
        return base.VisitAttributeList(node);
    }
}

/// The method calling the Syntax Rewriter
private static async Task<bool> ModifySolutionUsingSyntaxRewriter(string solutionPath)
{
    using (var workspace = MSBuildWorkspace.Create())
    {
        // Selects a Solution File
        var solution = await workspace.OpenSolutionAsync(solutionPath);
        // Iterates through every project
        foreach (var project in solution.Projects)
        {
            // Iterates through every file
            foreach (var document in project.Documents)
            {
                // Selects the syntax tree
                var syntaxTree = await document.GetSyntaxTreeAsync();
                var root = syntaxTree.GetRoot();
```

```
                // Generates the syntax rewriter
                var rewriter = new AttributeStatementChanger();
                root = rewriter.Visit(root);

                // Exchanges the document in the solution by the newly generated document
                solution = solution.WithDocumentSyntaxRoot(document.Id, root);
            }
        }
        // applies the changes to the solution
        var result = workspace.TryApplyChanges(solution);
        return result;
    }
}
```

El ejemplo anterior se puede probar para la siguiente clase:

```
public class Program
{
    [PreviousAttribute()]
    static void Main(string[] args)
    {
    }
}
```

Lea Cambiar el código fuente con Roslyn en línea:
https://riptutorial.com/es/roslyn/topic/5221/cambiar-el-codigo-fuente-con-roslyn

Capítulo 5: Modelo semantico

Introducción

En contraste con la API Syntax, que expone todo tipo de información de nivel de sintaxis, el modelo semántico le da a nuestro código más "significado" y nos permite responder preguntas como "¿Qué nombres están en el alcance en esta ubicación?", "¿A qué miembros se puede acceder desde ¿Este método? "," ¿Qué variables se utilizan en este bloque de texto? "," ¿A qué se refiere este nombre / expresión? ".

Observaciones

- Consultar el modelo semántico es más costoso que consultar el árbol de sintaxis, debido al hecho de que lo más común es que desencadene una compilación.

Examples

Obtención del modelo semántico

Existen varias formas de obtener el modelo semático.

- De una clase de `Document`

```
Document document = ...;
SemanticModel semanticModel = await document.GetSemanticModelAsync();
```

- De una clase de `Compilation`

```
CSharpCompilation compilation = ...;
var semanticModel = await compilation.GetSemanticModel(syntaxTree);
```

- Desde un `AnalysisContext` . Por ejemplo, dentro de un `DiagnosticAnalyzer` puedes hacer:

```
public override void Initialize(AnalysisContext context)
{
    context.RegisterSemanticModelAction(x =>
    {
        var semanticModel = x.SemanticModel;
        // Do magical magic here.
    });
}
```

Consigue todas las referencias a un método.

```
var syntaxRoot = await document.GetSyntaxRootAsync();
```

```
var semanticModel = await document.GetSemanticModelAsync();
var sampleMethodInvocation = syntaxRoot
    .DescendantNodes()
    .OfType<InvocationExpressionSyntax>()
    .First();

var sampleMethodSymbol = semanticModel.GetSymbolInfo(sampleMethodInvocation).Symbol;
var referencesToSampleMethod = await SymbolFinder.FindReferencesAsync(sampleMethodSymbol,
document.Project.Solution);
```

Lea Modelo semantico en línea: https://riptutorial.com/es/roslyn/topic/9772/modelo-semantico

Capítulo 6: Usando espacios de trabajo

Introducción

El espacio de trabajo es una representación programática de la jerarquía de C # que consta de una solución, proyectos secundarios y documentos secundarios.

Observaciones

- Actualmente no hay un espacio de trabajo de MSBuild que admita proyectos compatibles con .NET Standard. Para más información ver aquí .

Examples

Creando un AdhocWorkspace y agregándole un nuevo proyecto y un archivo.

La idea detrás de `AdhocWorkspace` es crear un espacio de trabajo sobre la marcha.

```
var workspace = new AdhocWorkspace();

string projectName = "HelloWorldProject";
ProjectId projectId = ProjectId.CreateNewId();
VersionStamp versionStamp = VersionStamp.Create();
ProjectInfo helloWorldProject = ProjectInfo.Create(projectId, versionStamp, projectName,
projectName, LanguageNames.CSharp);
SourceText sourceText = SourceText.From("class Program { static void Main() {
System.Console.WriteLine(\"HelloWorld\"); } }");

Project newProject = workspace.AddProject(helloWorldProject);
Document newDocument = workspace.AddDocument(newProject.Id, "Program.cs", sourceText);
```

Crear un MSBuildWorspace, cargar una solución y obtener todos los documentos en toda esa solución

`MSBuildWorspace` se basa en el concepto de manejo de soluciones MSBuild (archivos `.sln`) y sus respectivos proyectos (`.csproj` , `.vbproj`). No se admite la adición de nuevos proyectos y documentos a este espacio de trabajo.

```
string solutionPath = @"C:\Path\To\Solution\Sample.sln";

MSBuildWorkspace workspace = MSBuildWorkspace.Create();
Solution solution = await workspace.OpenSolutionAsync(nancyApp);

var allDocumentsInSolution = solution.Projects.SelectMany(x => x.Documents);
```

Obtención de VisualStudioWorkspace desde una extensión de Visual Studio

A diferencia de los otros tipos de espacios de trabajo, `VisualStudioWorkspace` no se puede crear manualmente. Se puede acceder al construir una extensión de Visual Studio.

Cuando `[YourVSPackage]Package.cs` dentro de su proyecto de paquete de extensión, vaya al `[YourVSPackage]Package.cs` . Allí puedes adquirir el espacio de trabajo de dos maneras:

```
protected override void Initialize()
{
    // Additional code...

    var componentModel = (IComponentModel)this.GetService(typeof(SComponentModel));
    var workspace = componentModel.GetService<VisualStudioWorkspace>();
}
```

O utilizando MEF:

```
[Import(typeof(VisualStudioWorkspace))]
public VisualStudioWorkspace ImportedWorkspace { get; set; }
```

Un excelente video tutorial sobre `VisualStudioWorkspace` , se puede encontrar aquí .

Lea Usando espacios de trabajo en línea: https://riptutorial.com/es/roslyn/topic/9755/usando-espacios-de-trabajo

Creditos

S. No	Capítulos	Contributors
1	Empezando con Roslyn	Community, Teodor Kurtev
2	Analizar el código fuente con Roslyn	andyp, Michael Rätzel, SJP, Stefano d'Antonio
3	Árbol de sintaxis	Teodor Kurtev
4	Cambiar el código fuente con Roslyn	SJP, Teodor Kurtev
5	Modelo semantico	Teodor Kurtev
6	Usando espacios de trabajo	Teodor Kurtev