無料電子ブック

学習

roslyn

Free unaffiliated eBook created from
Stack Overflow contributors.

#roslyn

1: ロスリンをはじめよう

Roslynからめるには、をてください

- シンタックスツリーAPI
- セマンティックモデルAPI
- のトピックをここにしてください 。

Examples

インストールまたはセットアップ

Roslynをいめるには、のNuGetパッケージがです。

- CとVBコンパイラ - `Microsoft.Net.Compilers` 。 インストールするには、パッケージマネージャコンソールでのコマンドをします。

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

- APIとサービス - `Microsoft.CodeAnalysis` 。 インストールするには、パッケージマネージャコンソールでのコマンドをします。

  ```
  nuget install Microsoft.CodeAnalysis
  ```

さらに、それはつけることができる、.NETコンパイラプラットフォームSDKテンプレートをインストールするにはいです、ここ 。これはあなたをるでしょう

- アナライザ、CodeFixes、スタンドアロンツールのをにするCとVisual Basicのテンプレート 。
- Visual Studioのビジュアライザツール `View -> Other Windows -> Syntax Visualizer` 。 のコードのツリーをべるのににです。

そののツールとリソース

- ロザリンクォーター

サンプルCプログラムをAPIびしにするためのツール。ツールはここにあります 。

- ソースビューア

Roslynソースコードをにするは、 こちらをください 。

オンラインでロスリンをはじめようをむ https://riptutorial.com/ja/roslyn/topic/2905/ロスリンをはじめよう

2: Roslynでソースコードをする

Examples

Cでののイントロスペクティブ

1. しいコンソールアプリケーションをする
2. **NuGet**パッケージ`Microsoft.CodeAnalysis`する
3. `Microsoft.CodeAnalysis.MSBuild`、 `System.Linq`および`Microsoft.CodeAnalysis.CSharp.Syntax`インポートします。
4. `Main`メソッドにのコードをします。

```
// 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();
```

プロジェクトをすると、にされたがにされます。つまり、プロジェクトをし、そのにがつかったことをします。

Cでな "Hello World"アプリケーションをする

`Main`メソッドに1のしいコンソールアプリケーションをします`Console.WriteLine("Hello World")`

`.csproj`ファイルへのパスをえておき、のようにきえます。

しいコンソールアプリケーションをし、 `Microsoft.CodeAnalysis` NuGetパッケージをインストールして、のコードをしてください

```
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();
```

これにより、 **Hello World**プロジェクトのすべてのファイルとすべてのノードがされます。

VB.NETでな "Hello World"アプリケーションをする

`Main`メソッドに1のしいコンソールアプリケーションをします`Console.WriteLine("Hello World")`

`.vbproj`ファイルへのパスをえておき、のようにきえます。

しいコンソールアプリケーションをし、 `Microsoft.CodeAnalysis` NuGetパッケージをインストールして、のコードをしてください

```
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()
```

これにより、 **Hello World**プロジェクトのすべてのファイルとすべての ノード がされます。

Cのテキストからソースコードをする

```
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();
```

このでは、をしているテキストから`Main`メソッドをします。

'var'のをする

`var`をってされたののをするには、 `SemanticModel GetSymbolInfo()`をびします。 `MSBuildWorkspace`をしてのソリューションを`MSBuildWorkspace` 、プロジェクトとそのドキュメントをできます。ドキュメントをして`SyntaxRoot`と`SemanticModel`をし、 `VariableDeclarations`をして、のようなされたの`Type`のシンボルをします。

```
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..
    }
}
```

オンラインでRoslynでソースコードをするをむ https://riptutorial.com/ja/roslyn/topic/4712/roslynでソースコードをする

3: Roslynとソースコードをする

き

Roslynをソースコードにするな

- Roslynはです。 ReplaceNodesのようなメソッドをびすと、のノードをするのではなく、しいノードをします。これは、しているオブジェクトをにするがあります。

Examples

ツリーをして、Cのすべてのメソッドののをきえます

のスニペットは、ソリューションにして、 `ReplacementAttribute`というのAttributeで`PreviousAttribute`とばれるすべてのをきえます。このサンプルでは、ツリーをでし、をけるすべてのノードをきえます。

```
    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;
        }
    }
```

のは、のクラスでテストできます。

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

のノードをきえるには、Methode root.ReplaceNodeをしないでください。ツリーはなので、あなたはのオブジェクトにりんでいます。ののでのスニペットをしても、されるはられません。

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

`ReplaceNode`へののびしでは、しいルートがされます。ただし、 `existingAttributesList`のはのルートのルートにしており、このためにきえることはできません。これにより、のがきえられ、のはされずにります。これは、すべてのびしがしいツリーにしないノードでされるためです。

SyntaxRewriterをして、Cのすべてのメソッドののをきえます

のスニペットは、ソリューションにして "ReplacementAttribute"というで "PreviousAttribute"とばれるすべてのをきえます。サンプルは、でSyntaxRewriterをしてをします。

```
/// <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;
    }
}
```

のは、のクラスでテストできます。

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

オンラインでRoslynとソースコードをするをむ https://riptutorial.com/ja/roslyn/topic/5221/roslynとソースコードをする

4: セマンティックモデル

き

APIとはに、すべてののレベルをします。セマンティックモデルはコードをよりのあるものにし、「こののスコープにはどのようながりますか」、このメソッドは "、"このテキストブロックではどのようながされていますか "、"この/はをしていますか "

- セマンティックモデルのクエリは、ツリーをクエリするよりもコストがかかります。これは、コンパイルがもにトリガされるためです。

Examples

セマンティックモデルの

sematicモデルをるをいくつかします。

- `Document`クラスから

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

- `Compilation`クラスから

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

- `AnalysisContext`から。 あなたがうことができる`DiagnosticAnalyzer`の

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

メソッドへのをすべてする

```
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);
```

オンラインでセマンティックモデルをむ https://riptutorial.com/ja/roslyn/topic/9772/セマンティックモデル

5: ワークスペースの

き

ワークスペースは、ソリューション、プロジェクト、ドキュメントからなるCのプログラムです。

- 、.NETにしたプロジェクトをサポートするMSBuildワークスペースはありません。はこちらをください 。

Examples

AdhocWorkspaceをし、しいプロジェクトとファイルをします。

`AdhocWorkspace`にある`AdhocWorkspace`は、そのでワークスペースをすることです。

```
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);
```

MSBuildWorspaceをし、ソリューションをみみ、そのすべてのソリューションですべてのドキュメントをする

`MSBuildWorspace`は、MSBuildソリューション `.sln`ファイルとそれぞれのプロジェクト `.csproj`、`.vbproj` をするコンセプトに`MSBuildWorspace`てされています。このワークスペースにしいプロジェクトやドキュメントをすることはできません。

```
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);
```

Visual Studioエクステンションからの**VisualStudioWorkspace**の

のタイプのワークスペースとはに、 `VisualStudioWorkspace`はではできません。 Visual Studioをビ

ルドするときにアクセスできます。

パッケージプロジェクトのにあるときは、`[YourVSPackage]Package.cs`ファイルにします。ここでは、2つのでワークスペースをできます。

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

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

またはMEFをして

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

`VisualStudioWorkspace`にするらしいビデオチュートリアルがここにあります 。

オンラインでワークスペースのをむ https://riptutorial.com/ja/roslyn/topic/9755/ワークスペースの

6: ツリー

き

Roslynコンパイラのの1つは、Syntax APIです。これは、コンパイラがVisual BasicおよびCプログラムをするためにするツリーをします。

- は、Roslynコンパイラのにおけるです。

Examples

ドキュメントからシンタクスツリールートをする。

ワークスペースから`Document`クラスへのアクセスがにあるワークスペースの 、ツリーのルートににアクセスできます。

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

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

LINQをしたのトラバース

LINQをしてをにナビゲートすることができます。たとえば、`A`まるをつすべての`ClassDeclarationSyntax`ノードされたクラスをにできます。

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

またはをつすべてのクラスをする

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

CSharpSyntaxWalkerをしてシンタックスツリーをトラバースする

`CSharpSyntaxWalker`クラスは、Visitorパターンのボックスであり、をトラバースするためにできます。は、 `A`というでまるをつすべての`struct`をするSyntax Walkerのなです

```
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);
        }
    }
}
```

SyntaxWalkerは、のようにできます。

```
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}");
```

オンラインでツリーをむ https://riptutorial.com/ja/roslyn/topic/9765/ツリー

クレジット

S. No		Contributors
1	ロスリンをはじめよう	Community, Teodor Kurtev
2	Roslynでソースコードをする	andyp, Michael Rätzel, SJP, Stefano d'Antonio
3	Roslynとソースコードをする	SJP, Teodor Kurtev
4	セマンティックモデル	Teodor Kurtev
5	ワークスペースの	Teodor Kurtev
6	ツリー	Teodor Kurtev