Бесплатная электронная книга

учусь

roslyn

Free unaffiliated eBook created from
Stack Overflow contributors.

#roslyn

Около

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

глава 1: Начало работы с roslyn

замечания

Чтобы начать с Roslyn, взгляните на:

- API дерева синтаксиса
- API семантической модели
- *Добавьте еще несколько тем* .

Examples

Установка или настройка

Чтобы начать возиться с Roslyn, вам понадобятся следующие пакеты NuGet:

- Компиляторы C # и VB - `Microsoft.Net.Compilers` . Чтобы установить его, вы можете запустить следующую команду в консоли диспетчера пакетов:

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

- Языковые API и службы - `Microsoft.CodeAnalysis` . Чтобы установить его, вы можете запустить следующую команду в консоли диспетчера пакетов:

  ```
  nuget install Microsoft.CodeAnalysis
  ```

Кроме того, полезно установить Шаблоны SDK для платформы .NET Compiler Platform, которые можно найти здесь . Это поможет вам:

- Шаблоны для C # и Visual Basic, которые позволяют создавать Анализаторы, CodeFixes и автономные инструменты анализа.
- Инструмент синтаксического визуализатора Visual Studio (`View -> Other Windows -> Syntax Visualizer`), который чрезвычайно полезен для изучения дерева синтаксиса существующего кода.

Дополнительные инструменты и ресурсы

- Рослин-котировщик

Инструмент для преобразования примера программы C # в вызовы API дерева синтаксиса. Сам инструмент можно найти здесь .

- Расширенный просмотрщик источников

Простой способ просмотра исходного кода Roslyn можно найти здесь .

Прочитайте Начало работы с roslyn онлайн: https://riptutorial.com/ru/roslyn/topic/2905/начало-работы-с-roslyn

глава 2: Анализ исходного кода с Roslyn

Examples

Интроспективный анализ анализатора в C

1. Создание нового **консольного приложения**
2. Добавить пакет **NuGet** `Microsoft.CodeAnalysis`
3. Импортируйте пространства имен `Microsoft.CodeAnalysis.MSBuild` , `System.Linq` и `Microsoft.CodeAnalysis.CSharp.Syntax`
4. Напишите следующий пример кода в методе `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();
```

При запуске проекта вы увидите переменную, объявленную сверху, напечатанную на экране. Это означает, что вы успешно проанализировали проект и нашли в нем переменную.

Проанализируйте простое приложение «Hello World» в C

Создайте новое консольное приложение с одной строкой в методе `Main` :
`Console.WriteLine("Hello World")`

Запомните путь к файлу `.csproj` и замените его в примере.

Создайте новое **консольное приложение** и установите пакет `Microsoft.CodeAnalysis` NuGet и попробуйте следующий код:

```
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();
```

Это напечатает все файлы и все **узлы синтаксиса** в проекте **Hello World** .

Анализ простого приложения Hello World в VB.NET

Создайте новое консольное приложение с одной строкой в методе `Main` :
`Console.WriteLine("Hello World")`

Запомните путь к файлу `.vbproj` и замените его в примере.

Создайте новое **консольное приложение** и установите пакет `Microsoft.CodeAnalysis` NuGet и попробуйте следующий код:

```
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()
```

Это напечатает все файлы и все **узлы синтаксиса** в проекте **Hello World** .

Исправить исходный код из текста в 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();
```

В этом примере будет напечатан `Main` метод из текста, анализирующего синтаксис.

Получить тип 'var'

Чтобы получить фактический тип переменной, объявленной с помощью `var` , вызовите `GetSymbolInfo()` в `SemanticModel` . Вы можете открыть существующее решение с помощью

`MSBuildWorkspace` , затем перечислить его проекты и их документы. Используйте документ для получения `SyntaxRoot` и `SemanticModel` , затем найдите `VariableDeclarations` и `SyntaxRoot` символы для `Type` объявленной переменной следующим образом:

```
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..
    }
}
```

Прочитайте Анализ исходного кода с Roslyn онлайн:
https://riptutorial.com/ru/roslyn/topic/4712/анализ-исходного-кода-с-roslyn

глава 3: Дерево синтаксиса

Вступление

Одной из основных частей компилятора Roslyn является API синтаксиса. Он предоставляет деревья синтаксиса, используемые компиляторами для понимания программ Visual Basic и C #.

замечания

- Дерево синтаксиса представляет собой дерево разбора в контексте компилятора Roslyn.

Examples

Получение корня дерева синтаксиса из документа.

Если у вас уже есть доступ к вашему классу `Document` из рабочей области (с помощью Workspaces), вам легко получить доступ к корню вашего дерева синтаксиса.

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

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

Перемещение дерева синтаксиса с помощью LINQ

Вы можете легко перемещаться по дереву синтаксиса с помощью LINQ. Например, легко получить все узлы `ClassDeclarationSyntax` (объявленные классы), которые имеют имя, начинающееся с буквы `A` :

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

Или получить все классы, которые имеют атрибуты:

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

Перемещение дерева синтаксиса с помощью CSharpSyntaxWalker

Класс `CSharpSyntaxWalker` представляет собой нестандартную реализацию шаблона Visitor,

который мы можем использовать для перемещения нашего дерева синтаксиса. Вот простой пример Синтаксического Уолкера, который собирает все `struct` -s, у которых есть имя, начиная с буквы `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);
        }
    }
}
```

Мы можем использовать наш SyntaxWalker следующим образом:

```
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}");
```

Прочитайте Дерево синтаксиса онлайн: https://riptutorial.com/ru/roslyn/topic/9765/дерево-синтаксиса

глава 4: Измените исходный код на Roslyn

Вступление

Практические примеры использования Roslyn для преобразования исходного кода.

замечания

- Синтаксические деревья Roslyn неизменяемы. Вызывая метод типа ReplaceNodes, мы генерируем новый узел, а не изменяем существующий. Это требует, чтобы вы всегда меняли объект, над которым работали.

Examples

Заменить существующие атрибуты для всех методов в C #, используя дерево синтаксиса

Следующий фрагмент заменяет все атрибуты, называемые `PreviousAttribute` , атрибутом `ReplacementAttribute` для всего решения. Образец вручную ищет дерево синтаксиса и заменяет все затронутые узлы.

```
    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;
        }
    }
```

Вышеприведенный пример может быть протестирован для следующего класса:

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

Вы не должны использовать Methode root.ReplaceNode для замены нескольких узлов. Поскольку дерево является неизменным, вы будете работать на разных объектах. Использование следующего фрагмента в приведенном выше примере не даст ожидаемого результата:

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

Первый вызов `ReplaceNode` создаст новый корневой элемент. Однако элементы в `existingAttributesList` принадлежат другому корню (предыдущему корневому элементу) и не могут быть заменены из-за этого. Это приведет к замене первого Атрибута, и следующие атрибуты останутся неизменными, поскольку все последовательные вызовы будут выполняться на узле, не присутствующем в новом дереве.

Замените существующие атрибуты для всех методов на C # с помощью SyntaxRewriter

Следующий фрагмент заменяет все атрибуты, называемые «PreviousAttribute» атрибутом « ReplacementAttribute» для всего решения. В образце вручную используется SyntaxRewriter для обмена атрибутами.

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

Вышеприведенный пример может быть протестирован для следующего класса:

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

Прочитайте Измените исходный код на Roslyn онлайн:
https://riptutorial.com/ru/roslyn/topic/5221/измените-исходный-код-на-roslyn

глава 5: Использование рабочих пространств

Вступление

Рабочая область представляет собой программное представление иерархии C #, которая состоит из решения, дочерних проектов и дочерних документов.

замечания

- В настоящее время нет рабочего пространства MSBuild, поддерживающего проекты, совместимые с стандартом .NET. Для получения дополнительной информации см. Здесь .

Examples

Создание AdhocWorkspace и добавление к нему нового проекта и файла.

Идея `AdhocWorkspace` заключается в том, чтобы создать рабочее пространство «на лету».

```
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);
```

Создание MSBuildWorspace, загрузка решения и получение всех документов во всем этом решении

`MSBuildWorspace` построена на основе концепции обработки решений MSBuild (`.sln` файлов) и их соответствующих проектов (`.csproj` , `.vbproj`). Добавление новых проектов и документов в это рабочее пространство не поддерживается.

```
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);
```

Получение VisualStudioWorkspace внутри расширения Visual Studio

В отличие от других типов рабочих пространств, `VisualStudioWorkspace` не может создаваться вручную. Доступ к нему можно получить при создании расширения Visual Studio.

Когда внутри проекта пакета расширения, перейдите в `[YourVSPackage]Package.cs` файл `[YourVSPackage]Package.cs` . Там вы можете приобрести рабочее пространство двумя способами:

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

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

Или с помощью MEF:

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

Отличный видеоурок о `VisualStudioWorkspace` , можно найти здесь .

Прочитайте Использование рабочих пространств онлайн:
https://riptutorial.com/ru/roslyn/topic/9755/использование-рабочих-пространств

глава 6: Семантическая модель

Вступление

В отличие от API синтаксиса, который предоставляет все виды информации о синтаксическом уровне, семантическая модель дает нам более «смысл» нашего кода и позволяет нам отвечать на такие вопросы, как «Какие имена находятся в области действия в этом месте?», «Что из участников доступно из этот метод? »,« Какие переменные используются в этом блоке текста? »,« Что означает это имя / выражение? ».

замечания

- Запрос семантической модели более дорогостоящий, чем запрос дерева синтаксиса, из-за того, что он чаще всего запускает компиляцию.

Examples

Получение семантической модели

Там есть способы получить семантическую модель.

- Из класса `Document`

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

- Из класса `Compilation`

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

- Из `AnalysisContext` . Пример Fro внутри `DiagnosticAnalyzer` вы можете сделать:

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

Получить все ссылки на метод

```
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);
```

Прочитайте Семантическая модель онлайн: https://riptutorial.com/ru/roslyn/topic/9772/семантическая-модель

кредиты

S. No	Главы	Contributors
1	Начало работы с roslyn	Community, Teodor Kurtev
2	Анализ исходного кода с Roslyn	andyp, Michael Rätzel, SJP, Stefano d'Antonio
3	Дерево синтаксиса	Teodor Kurtev
4	Измените исходный код на Roslyn	SJP, Teodor Kurtev
5	Использование рабочих пространств	Teodor Kurtev
6	Семантическая модель	Teodor Kurtev