**EBook Gratuito**

APPRENDIMENTO

# roslyn

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#roslyn

# Sommario

# Di

# Capitolo 1: Iniziare con Roslyn

## Osservazioni

Per iniziare con Roslyn, dai un'occhiata a:

- Syntax Tree API
- API del modello semantico
- *Aggiungi altri argomenti qui* .

## Examples

### Installazione o configurazione

Per iniziare a armeggiare con Roslyn avrai bisogno dei seguenti pacchetti NuGet:

- I compilatori C # e VB - `Microsoft.Net.Compilers` . Per installarlo è possibile eseguire il seguente comando nella Console di Gestione pacchetti:

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

- Le API e i servizi linguistici - `Microsoft.CodeAnalysis` . Per installarlo è possibile eseguire il seguente comando nella Console di Gestione pacchetti:

  ```
  nuget install Microsoft.CodeAnalysis
  ```

Inoltre, è utile installare i modelli di .NET Platform SDK Compiler, che possono essere trovati qui . Questo ti porterà:

- Modelli per C # e Visual Basic che consentono la creazione di Analizzatori, CodeFix e strumenti di analisi autonomi.
- Lo strumento Sintassi Visualizer per Visual Studio ( `View -> Other Windows -> Syntax Visualizer` ), che è estremamente utile per esaminare l'albero di sintassi del codice esistente.

### Ulteriori strumenti e risorse

- Il quoter di Roslyn

Uno strumento per la conversione di un programma C # di esempio per la sintassi delle chiamate API dell'albero. Lo strumento stesso può essere trovato qui .

- Visualizzatore sorgente avanzato

Un modo semplice per visualizzare il codice sorgente Roslyn può essere trovato qui .

Leggi Iniziare con Roslyn online: https://riptutorial.com/it/roslyn/topic/2905/iniziare-con-roslyn

# Capitolo 2: Albero della sintassi

## introduzione

Una delle parti principali del compilatore di Roslyn è l'API Syntax. Espone gli alberi di sintassi utilizzati dai compilatori per comprendere i programmi Visual Basic e C #.

## Osservazioni

- Syntax Tree è un Parse Tree nel contesto del compilatore di Roslyn.

## Examples

### Ottenere il Syntax Tree Root da un documento.

Se hai già accesso alla tua classe `Document` dal tuo spazio di lavoro ( Usando Aree di lavoro ) è facile accedere alla radice dell'albero Syntax.

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

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

### Attraversamento dell'albero della sintassi Utilizzando LINQ

Puoi facilmente navigare in un albero della sintassi usando LINQ. Ad esempio è facile ottenere tutti i nodi `ClassDeclarationSyntax` (classi dichiarate), che hanno un nome che inizia con la lettera `A` :

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

O ottenendo tutte le classi che hanno attributi:

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

### Attraversamento dell'albero della sintassi mediante CSharpSyntaxWalker

La classe `CSharpSyntaxWalker` è fuori dall'implementazione del pattern Visitor, che possiamo usare per attraversare il nostro Syntax Tree. Ecco un semplice esempio di Syntax Walker che raccoglie tutte le `struct` -s che hanno un nome, iniziando con la lettera `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);
        }
    }
}
```

Possiamo usare il nostro SyntaxWalker nel modo seguente:

```
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}");
```

Leggi Albero della sintassi online: https://riptutorial.com/it/roslyn/topic/9765/albero-della-sintassi

# Capitolo 3: Analizza il codice sorgente con Roslyn

## Examples

### Analisi introspettiva di un analizzatore in C #

1. Crea una nuova **applicazione console**
2. Aggiungere il pacchetto **NuGet** `Microsoft.CodeAnalysis`
3. Importare i namespace `Microsoft.CodeAnalysis.MSBuild` , `System.Linq` e `Microsoft.CodeAnalysis.CSharp.Syntax`
4. Scrivi il seguente codice di esempio nel metodo `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();
```

Durante l'esecuzione del progetto, vedrai la variabile dichiarata in alto stampata sullo schermo. Ciò significa che hai analizzato correttamente un progetto e trovato una variabile in esso.

### Analizza una semplice applicazione "Hello World" in C #

Creare una nuova applicazione console con una riga nel metodo `Main` : `Console.WriteLine("Hello`

`World")`

Ricorda il percorso del file `.csproj` e sostituiscilo nell'esempio.

Creare una nuova **applicazione console** e installare il pacchetto NuGet `Microsoft.CodeAnalysis` e provare il seguente codice:

```
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();
```

Questo stamperà tutti i file e tutti i **nodi di sintassi** nel progetto **Hello World** .

## Analizza una semplice applicazione "Hello World" in VB.NET

Creare una nuova applicazione console con una riga nel metodo `Main` : `Console.WriteLine("Hello World")`

Ricordare il percorso del file `.vbproj` e sostituirlo nell'esempio.

Creare una nuova **applicazione console** e installare il pacchetto NuGet `Microsoft.CodeAnalysis` e provare il seguente codice:

```
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()
```

Questo stamperà tutti i file e tutti i **nodi di sintassi** nel progetto **Hello World** .

## Analizzare il codice sorgente dal testo in 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();
```

Questo esempio stamperà il metodo `Main` dal testo che analizza la sintassi.

## Ottieni il tipo di 'var'

Per ottenere il tipo effettivo per una variabile dichiarata utilizzando `var` , chiamare `GetSymbolInfo()` sul `SemanticModel` . È possibile aprire una soluzione esistente utilizzando `MSBuildWorkspace` , quindi enumerarne i progetti e i relativi documenti. Utilizzare un documento per ottenere la sua `SyntaxRoot` e `SemanticModel` , quindi cercare `VariableDeclarations` e recuperare i simboli per il `Type` di una variabile dichiarata in questo modo:

```
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..
    }
}
```

Leggi Analizza il codice sorgente con Roslyn online:
https://riptutorial.com/it/roslyn/topic/4712/analizza-il-codice-sorgente-con-roslyn

# Capitolo 4: Cambia il codice sorgente con Roslyn

## introduzione

Esempi pratici di utilizzo di Roslyn per le trasformazioni del codice sorgente.

## Osservazioni

- Gli alberi di sintassi di Roslyn sono immutabili. Chiamando un metodo come ReplaceNodes, generiamo un nuovo nodo anziché modificare quello esistente. Ciò richiede di cambiare sempre l'oggetto su cui hai lavorato.

## Examples

### Sostituisci gli attributi esistenti per tutti i metodi in C # usando la struttura della sintassi

Il seguente snippet sostituisce tutti gli attributi chiamati `PreviousAttribute` da un attributo denominato `ReplacementAttribute` per un'intera soluzione. L'esempio ricerca manualmente l'albero della sintassi e sostituisce tutti i nodi interessati.

```
    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;
        }
    }
```

L'esempio sopra può essere testato per la seguente classe:

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

Non si dovrebbe utilizzare il metodo root.ReplaceNode per sostituire più nodi. Poiché l'albero è immutabile, lavorerai su diversi oggetti. L'utilizzo del seguente frammento nell'esempio sopra non produrrebbe il risultato atteso:

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

La prima chiamata a `ReplaceNode` creerebbe un nuovo elemento radice. Tuttavia, gli elementi in `existingAttributesList` appartengono a una radice diversa (l'elemento radice precedente) e non possono essere sostituiti a causa di ciò. Ciò comporterebbe la sostituzione del primo Attributo e i seguenti Attributi rimasti invariati poiché tutte le chiamate consecutive verrebbero eseguite su un nodo non presente nella nuova struttura.

## Sostituisci gli attributi esistenti per tutti i metodi in C # usando un

## SyntaxRewriter

Il seguente frammento sostituisce tutti gli Attributi denominati "PreviousAttribute" con un Attributo chiamato "ReplacementAttribute" per un'intera soluzione. L'esempio utilizza manualmente un SyntaxRewriter per scambiare gli attributi.

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

L'esempio sopra può essere testato per la seguente classe:

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

Leggi Cambia il codice sorgente con Roslyn online:
https://riptutorial.com/it/roslyn/topic/5221/cambia-il-codice-sorgente-con-roslyn

# Capitolo 5: Modello semantico

## introduzione

A differenza dell'API Syntax, che espone tutti i tipi di informazioni sul livello di sintassi, il modello semantico fornisce al nostro codice più "significato" e ci consente di rispondere a domande come "Quali sono i nomi in questo campo?", "Quali membri sono accessibili da questo metodo? "," Quali variabili sono usate in questo blocco di testo? "," A cosa si riferisce questo nome / espressione? ".

## Osservazioni

- Interrogare il modello semantico è più costoso rispetto a eseguire una query sull'albero della sintassi, poiché in genere genera una compilazione.

## Examples

### Ottenere il modello semantico

Ci sono alcuni modi per ottenere il modello sematico.

- Da una classe di `Document`

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

- Da una classe di `Compilation`

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

- Da un `AnalysisContext` . Per esempio in un `DiagnosticAnalyzer` puoi fare:

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

### Ottieni tutti i riferimenti a un metodo

```
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);
```

Leggi Modello semantico online: https://riptutorial.com/it/roslyn/topic/9772/modello-semantico

# Capitolo 6: Utilizzo di aree di lavoro

## introduzione

L'area di lavoro è una rappresentazione programmatica della gerarchia C # che consiste in una soluzione, progetti figlio e documenti figlio.

## Osservazioni

- Attualmente non esiste un'area di lavoro di MSBuild che supporti progetti conformi allo standard .NET. Per maggiori informazioni vedi qui .

## Examples

### Creare un AdhocWorkspace e aggiungervi un nuovo progetto e un file.

L'idea alla base di `AdhocWorkspace` è creare uno spazio di lavoro al volo.

```
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);
```

### Creare uno spazio MSBuildWors, caricare una soluzione e ottenere tutti i documenti in quella soluzione

`MSBuildWorspace` è costruito attorno al concetto di gestione delle soluzioni MSBuild (file `.sln` ) e dei rispettivi progetti ( `.csproj` , `.vbproj` ). L'aggiunta di nuovi progetti e documenti a questo spazio di lavoro non è supportata.

```
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);
```

### Ottenere il VisualStudioWorkspace dall'interno di un'estensione di Visual Studio

A differenza degli altri tipi di aree di lavoro, `VisualStudioWorkspace` non può essere creato manualmente. È possibile accedervi quando si crea un'estensione di Visual Studio.

Quando ci si trova all'interno del progetto del pacchetto di estensione, andare al file `[YourVSPackage]Package.cs` . Lì puoi acquisire lo spazio di lavoro in due modi:

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

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

O usando MEF:

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

Un ottimo video tutorial su `VisualStudioWorkspace` , può essere trovato qui .

Leggi Utilizzo di aree di lavoro online: https://riptutorial.com/it/roslyn/topic/9755/utilizzo-di-aree-di-lavoro

# Titoli di coda

| S. No | Capitoli | Contributors |
|---|---|---|
| 1 | Iniziare con Roslyn | Community, Teodor Kurtev |
| 2 | Albero della sintassi | Teodor Kurtev |
| 3 | Analizza il codice sorgente con Roslyn | andyp, Michael Rätzel, SJP, Stefano d'Antonio |
| 4 | Cambia il codice sorgente con Roslyn | SJP, Teodor Kurtev |
| 5 | Modello semantico | Teodor Kurtev |
| 6 | Utilizzo di aree di lavoro | Teodor Kurtev |