原文地址：

https://docs.microsoft.com/zh-cn/dotnet/csharp/roslyn-sdk/get-started/syntax-analysis

**安装 .NET Compiler Platform SDK**

安装 .NET Compiler Platform SDK 我们可以在Visual Studio查看语法树

执行如下步骤安装 .NET Compiler Platform SDK

运行“Visual Studio 安装程序”

选择“修改”

检查“Visual Studio 扩展开发”工作负荷 。

在摘要树中打开“Visual Studio 扩展开发”节点 。

选中“.NET Compiler Platform SDK”框 。 将在可选组件最下面找到它。

**安装包**

使用Roslyn前先安装包 Microsoft.CodeAnalysis.CSharp

**了解语法树**

语法树由生成项目时或开发人员按 F5 时所运行的分析程序生成。

语法树的四个主要构建基块为：

Microsoft.CodeAnalysis.SyntaxTree 类：

代表整个分析树。 SyntaxTree 是一种带有语言特定派生类的抽象类。 使用 Microsoft.CodeAnalysis.CSharp.CSharpSyntaxTree类对 C# 的文本进行分析。

Microsoft.CodeAnalysis.SyntaxNode 类：

表示语法节点，如声明、语句、子句和表达式等语法

Microsoft.CodeAnalysis.SyntaxToken 结构：

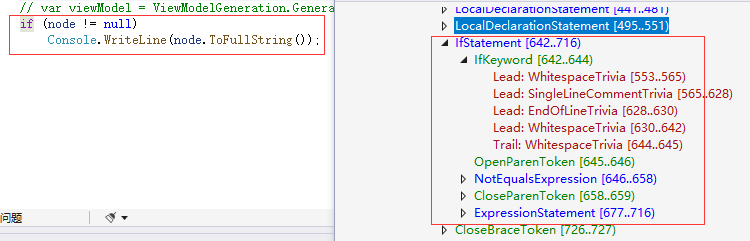
表示语法标志，它代表独立的关键词、标识符、运算符或标点。

Microsoft.CodeAnalysis.SyntaxTrivia 结构：

它代表语法上不重要的信息，例如标记、预处理指令和注释之间的空格。

在 Visual Studio 中，选择“视图” > “其他窗口” > “语法可视化工具”查看代码语法

SyntaxNode：蓝色 | SyntaxToken：绿色 | SyntaxTrivia：红色



**遍历语法节点**

static void Main(string[] args)

{

const string programText =

@"using System;

using System.Collections;

using System.Linq;

using System.Text;

namespace HelloWorld

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine(""Hello, World!"");

}

}

}";

SyntaxTree tree = CSharpSyntaxTree.ParseText(programText);

// 获取根语法节点

CompilationUnitSyntax root = tree.GetCompilationUnitRoot();

// 语法的类型，当前为 CompilationUnit 类型

Console.WriteLine($"The tree is a {root.Kind()} node.");

// 遍历语法使用的 Using

foreach (UsingDirectiveSyntax element in root.Usings)

Console.WriteLine($"\t{element.Name}");

// 遍历根语法节点的成员，返回子语法节点

MemberDeclarationSyntax firstMember = root.Members[0];

// 查看语法节点的类型

Console.WriteLine($"The first member is a {firstMember.Kind()}.");

// 其实该子语法节点是个 命名空间语法节点，所以强制转换

var helloWorldDeclaration = (NamespaceDeclarationSyntax)firstMember;

// 获取子语法节点

var programDeclaration = (ClassDeclarationSyntax)helloWorldDeclaration.Members[0];

// Identifier 是语法标志，包含类的名称

Console.WriteLine($"{programDeclaration.Identifier} class.");

// 获取子语法节点

var mainDeclaration = (MethodDeclarationSyntax)programDeclaration.Members[0];

// 返回类型

Console.WriteLine($"The return type is {mainDeclaration.ReturnType}.");

// 遍历参数语法节点

foreach (ParameterSyntax item in mainDeclaration.ParameterList.Parameters)

Console.WriteLine($"The type of the {item.Identifier} parameter is {item.Type}.");

var argsParameter = mainDeclaration.ParameterList.Parameters[0];

Console.ReadKey();

}

**查询方法**

除了遍历树，还可以使用 Microsoft.CodeAnalysis.SyntaxNode 上定义的查询方法来探索语法树

SyntaxTree tree = CSharpSyntaxTree.ParseText(programText);

// 获取根语法节点

CompilationUnitSyntax root = tree.GetCompilationUnitRoot();

// DescendantNodes 为获取所有子语法节点（Descendant：后代）

var firstParameters = from methodDeclaration in root.DescendantNodes()

.OfType<MethodDeclarationSyntax>()

where methodDeclaration.Identifier.ValueText == "Main"

select methodDeclaration.ParameterList.Parameters.First();

var argsParameter2 = firstParameters.Single();

Console.WriteLine(argsParameter2.ToFullString());

**语法查看器（CSharpSyntaxWalker）**

查看器接收一个语法节点，并遍历该节点和其子节点

定义查看器

public class UsingCollector : CSharpSyntaxWalker

{

public ICollection<UsingDirectiveSyntax> Usings { get; } = new List<UsingDirectiveSyntax>();

public override void VisitUsingDirective(UsingDirectiveSyntax node)

{

if (node.Name.ToString() == "System" || node.Name.ToString().StartsWith("System."))

{

this.Usings.Add(node);

}

}

}

使用查看器遍历语法数

SyntaxTree tree = CSharpSyntaxTree.ParseText(programText);

// 获取根语法节点

CompilationUnitSyntax root = tree.GetCompilationUnitRoot();

var collector = new UsingCollector();

collector.Visit(root);

foreach (var directive in collector.Usings)

{

Console.WriteLine(directive.Name);

}