Roslyn code analyzers

Sergey Ogorodnikov
C# Developer

One Inc
2/7/2019

Plan

- Get Started
- Write Code Analyzer
- Use standalone tool
- Go live!
- Write Code Fixes

Get Started

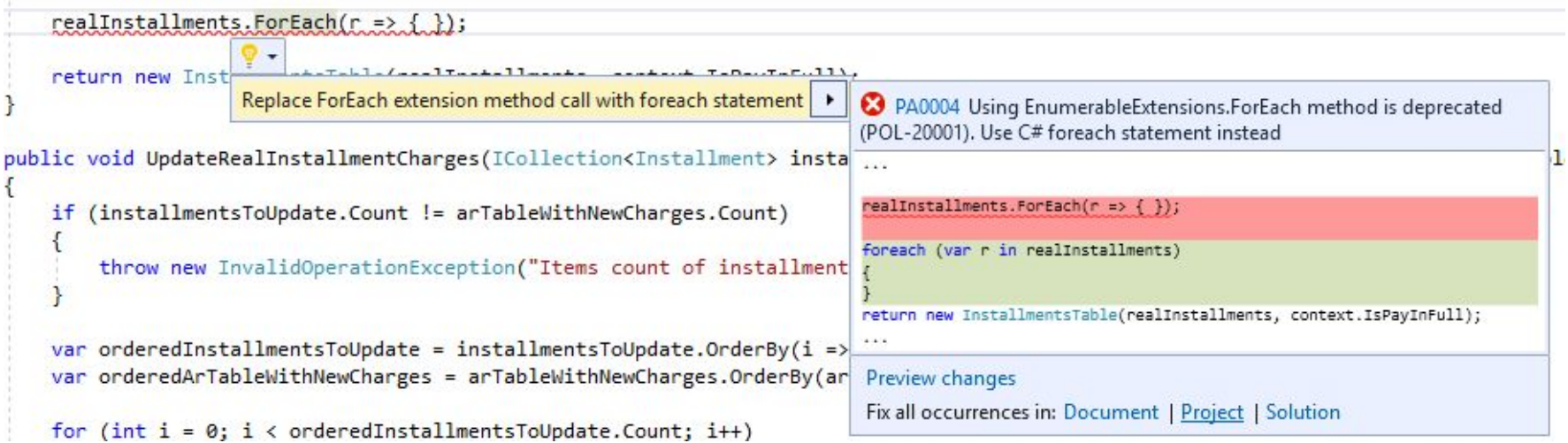
Roslyn is a new (~2015) compiler platform with open APIs:

- Compiler API
- Diagnostics API
- Workspaces API

It allows to create code analyzers/fixes and standalone tools. Code analyzers are used in build pipeline and in Visual Studio.

Open source: https://github.com/dotnet/roslyn

Get Started


realInstallments.ForEach(r => { });
return new Inst
}
Replace ForEach extension method call with foreach statement
PA0004 Using EnumerableExtensions.ForEach method is deprecated (POL-20001). Use C# foreach statement instead
...
realInstallments.ForEach(r => { });
foreach (var r in realInstallments)
{
}
return new InstallmentsTable(realInstallments, context.IsPayInFull);
...
Preview changes
Fix all occurrences in: Document | Project | Solution
public void UpdateRealInstallmentCharges(ICollection<Installment> insta
{
if (installmentsToUpdate.Count != arTableWithNewCharges.Count)
{
throw new InvalidOperationException("Items count of installment
}
var orderedInstallmentsToUpdate = installmentsToUpdate.OrderBy(i =>
var orderedArTableWithNewCharges = arTableWithNewCharges.OrderBy(ar
for (int i = 0; i < orderedInstallmentsToUpdate.Count; i++)

Get Started

Visual Studio 2015+

Install the .NET Compiler Platform ("Roslyn") SDK

Start with a template code analyzer solution:

- Portable/netstandard library with sample code analyzer & code fix (MEF!)
- Tests with some base classes
- VSIX to create a VS extension

Write Code Analyzer

```
namespace OneInc.PolicyOne.Analyzers.CodeAnalyzer
{
    [DiagnosticAnalyzer(LanguageNames.CSharp)]
    internal class ForbidRegexUsageAnalyzer : DiagnosticAnalyzer
    {
        private readonly DiagnosticDescriptor _descriptor = new DiagnosticDescriptor(
            "PA0007",
            "Regex is an EVIL!",
            "Please don't use regex!",
            "Code quality",
            DiagnosticSeverity.Error,
            true);

        public override ImmutableArray<DiagnosticDescriptor> SupportedDiagnostics => ImmutableArray.Create(_descriptor);

        public override void Initialize(AnalysisContext context)
        {
            var a = new Regex(" ");
            context.RegisterSyntaxNodeAction(AnalyzeObjectCreation, SyntaxKind.ObjectCreationExpression);
        }

        private void AnalyzeObjectCreation(SyntaxNodeAnalysisContext ctx)
        {
            var creationExpression = (ObjectCreationExpressionSyntax)ctx.Node;
            var classFullName = creationExpression.Type.GetClassFullName();

            if (classFullName == "System.Text.RegularExpressions.Regex")
            {
                ctx.ReportDiagnostic(Diagnostic.Create(_descriptor, creationExpression.GetLocation()));
            }
        }
    }
}
```

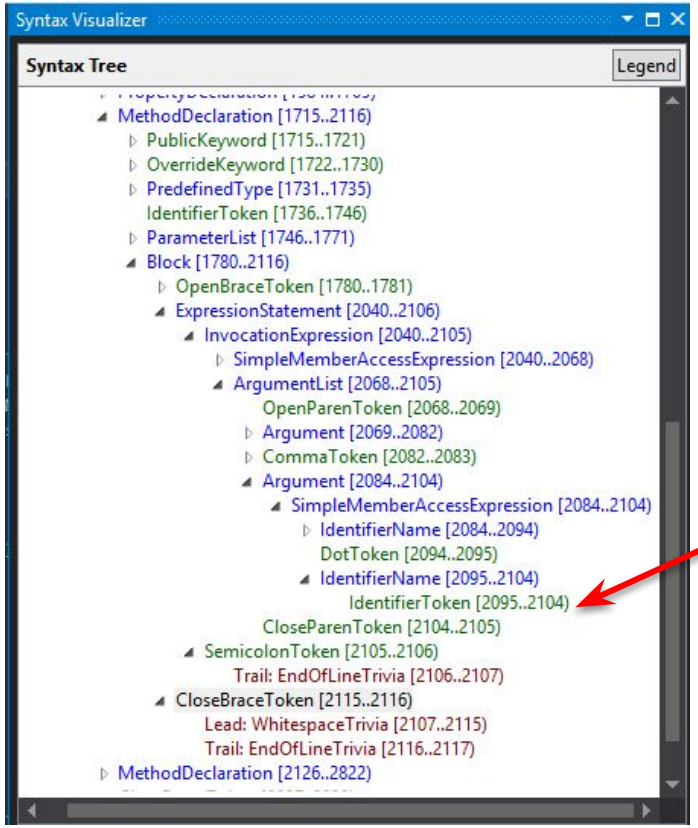
SyntaxNode

Document has syntax tree with root and childrens of type SyntaxNode

SyntaxKind - Enum with 485 items:

- 29 *Statement
- 93 *Expression
- 28 *Trivia
- 80 *Token
- 130 *Keyword

Use Syntax Visualizer, Luke!




```
public override void Initialize(AnalysisContext context)
{
    context.RegisterSymbolAction(AnalyzeSymbol, SymbolKind.NamedType);
}
```

Properties

Type SyntaxToken
Kind CloseBraceToken

ContainsAnnotations	False
ContainsDiagnostics	False
ContainsDirectives	False
FullSpan	[2107..2117)
HasLeadingTrivia	True
HasStructuredTrivia	False
HasTrailingTrivia	True
IsMissing	False
Language	C#
LeadingTrivia	
Parent	{ // TODO: Consider registering other a
RawKind	8206
Span	[2115..2116)
SpanStart	2115
SyntaxTree	using System;using System.Collections.Gene
Text	}
TrailingTrivia	
Value	}
ValueText	}

Semantic model

Available only if document compilation succeeded

- ctx.SemanticModel.GetSymbolInfo(constuctorInitializeNode);
- ctx.SemanticModel.GetDeclaredSymbol(ctx.Node);
- ctx.SemanticModel.GetTypeInfo(expressionNode);
- var flow = ctx.SemanticModel.AnalyzeControlFlow(statement);
- var flow = ctx.SemanticModel.AnalyzeDataFlow(statement);

TDD in action

```
        [Test]
        public void CodeFix_WhenSourceHasForEachMethodCallAndLambdaIsBlock_ShouldReplaceWithForeachStastement()
        {
            const string source = @"
using System;
using System.Linq;
using OneInc.PolicyOne.Common.Collections;
class A {
    public void Run() {
        Console.WriteLine(""Hello world!"");
        Enumerable.Range(1, 5).ForEach(i => { Console.WriteLine(i); });
    }
}
" + ForEachSourceCode;

            const string fixedSource = @"
using System;
using System.Linq;
using OneInc.PolicyOne.Common.Collections;
class A {
    public void Run() {
        Console.WriteLine(""Hello world!"");
        foreach (var i in Enumerable.Range(1, 5))
        {
            Console.WriteLine(i);
        }
    }
}
" + ForEachSourceCode;

            VerifyCSharpFix(source, fixedSource, allowNewCompilerDiagnostics: true);
        }
```

Standalone tool

Allows to see your analyzer in action without publishing to local feed. Can be useful to collect rule exceptions list.

- Prepare settings file
- Run your analyzer on OneInc.PolicyOne.All solution
- See if there are any AD0001 errors
- Try broke your code to ensure it really provides diagnostics
- Measure performance

Go live!

Create Nuget package

Install to project (will run some powershell magic)

Run compile (msbuild will use referenced library and will produce warns/errors)

! Look for AD0001 warnings !

Or create vsix extension to install analyzers machine wide

P1 Specific

- Our own base Code Analyzer class which provides settings and error processing
- 9 Code Analyzers
- 2 Code Fixes (ForEach)
- Console tool to run analyzer over All solution
- One settings file for all P1 analyzers
- Core 1.x projects problems

P1 Analyzers list

- PA0001 - Forbid direct interface implementation + exceptions list (as is)
- PA0002 - use methods in pair + exceptions list (tests)
- PA0003 - forbid directive #if debug
- PA0004 - forbid foreach extension method (+ Code Fix)
- PA0005 - forbid specified attributes usage (debug)
- PA0006 - forbid Debug.Assert() usage
- PA0007 - ensure builder calls chain ended with final method
- PA0008 - NonAbstractClassShouldBeSealed (+ 2 Code Fix)
- PA0009 - dependency (namespaces) white/black list

Write Code Fix

```
[ExportCodeFixProvider(LanguageNames.CSharp, Name = nameof(ForbidForEachExtensionUsageAnalyzer))]
[Shared]
public class ForbidForEachExtensionUsageCodeFixProvider : CodeFixProvider
{
    public override ImmutableArray<string> FixableDiagnosticIds { get; } =
        ImmutableArray.Create(ForbidForEachExtensionUsageAnalyzer.DiagnosticId);

    public override FixAllProvider GetFixAllProvider()
    {
        return WellKnownFixAllProviders.BatchFixer;
    }

    public override async Task RegisterCodeFixesAsync(CodeFixContext context)
    {
        var root = await context.Document.GetSyntaxRootAsync(context.CancellationToken).ConfigureAwait(false);

        foreach (var diagnostic in context.Diagnostics)
        {
            var invocation = root.FindToken(diagnostic.Location.SourceSpan.Start)
                .Parent.AncestorsAndSelf()
                .OfType<InvocationExpressionSyntax>()
                .First(
                    call => call.GetCalledMethodName()?.Identifier.Text == ForbidForEachExtensionUsageAnalyzer.ForeachMethodShortName);

            context.RegisterCodeFix(
                CodeAction.Create(Title, c => ChangeToForeachStatement(context.Document, invocation, c), Title),
                diagnostic);
        }
    }

    private static async Task<Document> ChangeToForeachStatement(
        Document document,
        InvocationExpressionSyntax invocation,
        CancellationToken cancellationToken)
    {
        var generator = SyntaxGenerator.GetGenerator(document);

        var containingStatement = GetContainingStatement(invocation);

        var root = await document.GetSyntaxRootAsync(cancellationToken).ConfigureAwait(false);
        root = root.TrackNodes(containingStatement);
        var invocationStatementTracked = root.GetCurrentNode(containingStatement);
```

Complexity

- Syntax complexity
- Semantic model usage
- Not to much materials
- Defects (5000 Issues)
- Lack of functionality (MSBuildWorkspace)
- Roslyn libs version compatibility

```
public static SimpleNameSyntax GetCalledMethodName(this InvocationExpressionSyntax invocationExpressionSyntax)
{
    if (invocationExpressionSyntax.Expression is SimpleNameSyntax nameSyntax)
    {
        return nameSyntax;
    }

    if (invocationExpressionSyntax.Expression is MemberAccessExpressionSyntax memberAccess)
    {
        return memberAccess.Name;
    }

    if (invocationExpressionSyntax.Expression is MemberBindingExpressionSyntax memberBindingAccess)
    {
        return memberBindingAccess.Name;
    }

    if (invocationExpressionSyntax.Expression is InvocationExpressionSyntax)
    {
        return null;
    }

    if (invocationExpressionSyntax.Expression is ElementAccessExpressionSyntax)
    {
        return null;
    }

    throw new Exception(
        $"InvocationExpressionSyntaxExtensions.GetCalledMethodName: invocationExpressionSyntax.Expression syntax
}
```

Bonus content! Typescript

TsLint is extensible: https://palantir.github.io/tslint/develop/custom-rules/

Custom P1 rules:

- Forbid import dependency rule
- Forbid ambient dependency rule
- AOT compatible component declaration rule
- Forbid function call
- Forbidden words usage in context (by “namespaces”)
- Check member name (styling) (+ Code Fix)
- Forbid Date construction
- Forbid reexport

TypeScript

```typescript
class ForbidNewDateRuleWalker extends Lint.RuleWalker {

    protected visitCallExpression(node: ts.CallExpression): void {
        this._checkForNowDateUsing(node);
        super.visitCallExpression(node);
    }

    private _checkForNowDateUsing(node: ts.CallExpression): void {
        if (node.expression.getText() === "Date.now") {
            this._addFailure(node, "Date.now()");
        }
    }

    private _addFailure(node: ts.Node, usageTypeString: string): void {
        const start = node.getStart();
        const width = node.getWidth();
        const failureMessage = `Forbid usage '${usageTypeString}' for getting now date. Please, use DateTimeService
        this.addFailure(this.createFailure(start, width, failureMessage));
    }
}

export class Rule extends Lint.Rules.AbstractRule {
    public apply(sourceFile: ts.SourceFile): Lint.RuleFailure[] {
        return this.applyWithWalker(new ForbidNewDateRuleWalker(sourceFile, this.getOptions()));
    }
}
```

Thanks for your attention!

Questions?