Настройка .NET анализаторов

Идея

В данном туториале представлено описание анализаторов кода и процесс их настройки для проверки .NET кода на соответствие принятому соглашению по его оформлению. Для проверки совместно используются:

- стандартные средства анализа кода Visual Studio / ссылка на документацию
- утилита StyleCop Analyzers / ссылка на репозиторий

Конфигурация анализаторов сформирована таким образом, что при нарушении основных правил сборка проекта приводит к ошибкам.

Актуальную версию конфигурации можно найти в данном репозитории.

Окружение

При написании туториала использовались:

- Windows 10 Pro 21H2
- Microsoft Visual Studio Community 2022 (64-bit, en). Version: 17.3.4.
- Console Application. Target Framework: .NET 6.0
- StyleCopAnalyzers. Version: 1.1.118

Настройка проверки

1. После создания проекта в Visual Studio, убедиться, что стандартные анализаторы кода включены.
2. Установить версию правил “6.0” стандартных анализаторов Visual Studio.
3. Включить проверку кода стандартными анализаторами Visual Studio при сборке проекта.
4. Установить StyleCop.Analyzers версии 1.1.118.
5. Создать пустые файлы `.editorconfig`, `stylecop.json` и `stylecop.ruleset` в проекте.
6. Добавить путь к файлам `stylecop.json` и `stylecop.ruleset` в конфигурацию проекта.
7. Заполнить файлы `.editorconfig`, `stylecop.json` и `stylecop.ruleset` правилами в соответствии с актуальной версией конфигурационных файлов.

Стандартные анализаторы Visual Studio

Назначение

Анализаторы платформы .NET проверяют код на C# или Visual Basic на наличие проблем с качеством и стилем кода. Если анализатор обнаруживает нарушения правил, появляется сообщение в виде предложения, предупреждения или ошибки — в зависимости от того, как настроено каждое правило.

Нарушения, выявленные при анализе кода, отображаются с префиксом "CA" или "IDE", чтобы отличить их от ошибок компилятора.

Все правила разделены на группы: `Design`, `Documentation`, `Globalization`, `Interoperability`, `Maintainability`, `Naming`, `Performance`, `SingleFile`, `Reliability`, `Security`, `Style`, `Usage`, `CodeQuality`. Их описание можно посмотреть в документации.

Включение проверки

Начиная с версии .NET 5, анализаторы кода включены по умолчанию в .NET SDK, поэтому устанавливать их отдельно не требуется. Если проект ориентирован на другую платформу .NET (.NET Core, .NET Standard или .NET Framework), необходимо вручную включить анализаторы путем задания свойству `EnableNETAnalyzers` значения `True` (в конфигурационном файле `*.csproj`):

```
<PropertyGroup>
  . . .
  <EnableNETAnalyzers>True</EnableNETAnalyzers>
</PropertyGroup>
```

Чтобы полностью отключить анализ кода нужно задать этому же свойству значение `False`. Про другие варианты установки анализаторов написано в документации.

Аналогичную настройку можно произвести в Visual Studio: Project properties → Code Analysis → .NET analyzers → Enable .NET analyzers.

Чтобы задать версию правил, которые будут использоваться при проверке и обеспечить воспроизводимость результатов проверки, необходимо задать свойство `AnalysisLevel` в файле конфигурации (варианты настройки можно посмотреть здесь).

```
<PropertyGroup>
  . . .
  <AnalysisLevel>6.0</AnalysisLevel>
</PropertyGroup>
```

Аналогичную настройку можно произвести в Visual Studio: Project properties → Code Analysis → .NET analyzers → Analysis level.

Описание файла конфигурации

Правила перечисляются в конфигурационном файле `.editorconfig` и задаются в формате `<ключ> = <значение>`. Ключи — это правила или группа правил, к которым применяется настройка. Значение — это уровень важности правила:

- `error` — нарушение правила приводит к ошибке при сборке проекта
- `warning` — нарушение правила приводит к предупреждению при сборке проекта
- `suggestion` — при нарушении правила появляются информационные сообщения
- `silent` — нарушение правил будет невидимо для пользователя (отображается только в меню рефакторинга)
- `none` — отключение правила
- `default` — к данному правилу применяется настройка по умолчанию в соответствии с заданной версией правил.

Формат записей для отдельных правил выглядит следующим образом:

```
dotnet_diagnostic.<rule ID>.severity = <value>
```

Формат записей для группы правил:

```
dotnet_analyzer_diagnostic.category-<category name>.severity = <value>
```

Пример настройки:

```
# Применение правил для файлов с расширением .cs и .vb
[*.{cs,vb}]

# Отключение правила IDE0040
dotnet_diagnostic.IDE0040.severity = none

# При нарушении правил категории "Style" будут появляться предупреждения
dotnet_analyzer_diagnostic.category-Style.severity = warning
```

Правила также можно задавать в формате `dotnet_diagnostic.<rule ID> = <value>:<severity>`, однако в этом случае они будут восприниматься только Visual Studio, и не будут работать при сборке проекта.

Если в файле задано несколько конфигураций, которые относятся к одному и тому же правилу, то приоритет выбирается в следующем порядке:

- конфигурация правила по его ID имеет приоритет выше чем конфигурация группы правил (куда входит данное правило)
- конфигурация группы правил имеет приоритет выше, чем конфигурация для всех правил

Помимо правил, используемых для .NET кода, существует ряд общих настроек, которые можно добавить в `.editorconfig` (и которые будут использоваться в Visual Studio):

- `indent_style`, `indent_size`, `tab_width` — для настройки отступов
- `end_of_line` — какой символ использовать в конце строки
- `charset` — кодировка файлов
- `trim_trailing_whitespace` — нужно ли убрать пробелы из концов строк
- `insert_final_newline` — нужно ли вставлять пустую строку в конце файла
- `root` — настройка, которая должна быть на самом верху файла конфигурации, определяющая, нужно ли наследовать конфигурацию из файлов `.editorconfig` в родительских папках.

Правила в файле `.editorconfig` применяются ко всем файлам в папках и подпапках директории, где располагается файл конфигурации. Для кастомизации правил к конкретным файлам и папкам проекта используются заголовки типа `[*.cs]`. Таким образом, помещая записи под разными заголовками, можно определять правила для разных файлов в зависимости от расширения.

Файл `.editorconfig` можно создать вручную или с помощью Visual Studio с предустановленными настройками: Project → Add New Item → C# Items → General → editorconfig File (.NET) или editorconfig File (default).

Анализ стиля кода

Назначение

Анализ стиля кода (правила типа "IDExxxx") проверяет код с точки зрения соблюдения оформления в соответствии с принятым стилем.

По умолчанию проверка выполняется при работе в Visual Studio и используется в меню рефакторинга. Чтобы проверка выполнялась при сборке как в Visual Studio, так и при через консоль, необходимо задать свойство `EnforceCodeStyleInBuild`:

```
<PropertyGroup>
  . . .
  <EnforceCodeStyleInBuild>True</EnforceCodeStyleInBuild>
</PropertyGroup>
```

Аналогичную настройку можно произвести в Visual Studio: Project properties → Code Analysis → .NET analyzers → Enforce code style on build.

Настройка правил

Большая часть правил имеет несколько опций, позволяющих кастомизировать стиль кода. Например, для правила IDE0063 “Use simple using statement” существует опция `csharp_prefer_simple_using_statement`:

```
// csharp_prefer_simple_using_statement = true
using var a = b;

// csharp_prefer_simple_using_statement = false
using (var a = b) { }
```

Далее чтобы задать уровень важности правила (например, warning), необходимо также задать:

```
[*.{cs,vb}]
dotnet_diagnostic.IDE0063.severity = warning
```

Анализ качества кода

Назначение

Анализ качества кода (правила типа "CAxxxx") проверяет проект с точки зрения безопасности, производительности и организации кода. Проверка выполняется при сборке проекта.

Настройка правил

У правил качества могут быть отдельные свойства, которые задаются для ограничения области проверки. Формат записи правил:

```
# Задание свойства для отдельного правила
dotnet_code_quality.<RuleId>.<OptionName> = <OptionValue>

# Задание свойства для категории правил
dotnet_code_quality.<RuleCategory>.<OptionName> = <OptionValue>

# Задание свойства в целом
dotnet_code_quality.<OptionName> = <OptionValue>
```

В качестве `OptionName` могут быть следующие значения:

- `api_surface` — какую часть кода анализировать (`public`, `internal`, `private`, `all`), значение по умолчанию — `public`
- `exclude_async_void_methods` — игнорировать асинхронные методы, которые не возвращают значение (`true`, `false`), значение по умолчанию — `false`
- `exclude_single_letter_type_parameters` — игнорировать типы параметров, состоящие из одного символа (`true`, `false`), значение по умолчанию — `false`

- `output_kind` — какие файлы анализировать (`ConsoleApplication`, `DynamicallyLinkedLibrary`, `NetModule`, `WindowsApplication`, `WindowsRuntimeApplication`, `WindowsRuntimeMetadata`), по умолчанию анализируются все файлы
- `required_modifiers` — объекты с каким модификатором анализировать (`none`, `public`, `private`, и т. д.), значение по умолчанию определяется в каждом правиле отдельно
- `exclude_extension_method_this_parameter` — игнорировать `this` в методах расширения (`true`, `false`), значение по умолчанию — `false`
- `null_check_validation_methods` — обозначение методов проекта, которые проводят проверку параметров на “null” (значения — имена методов)
- `additional_string_formatting_methods` — обозначение методов проекта для форматирования строк (значения — имена методов)
- `excluded_type_names_with_derived_types` — игнорировать объекты по типу (значения — имена типов)
- `excluded_symbol_names` — игнорировать объекты по имени (возможные значения — имена объектов)
- `disallowed_symbol_names` — имена объектов, которые нельзя использовать (значения — имена объектов)

Помимо свойств, также необходимо задать важность правила. Пример конфигурации правила CA1720: Identifiers should not contain type names:

```
[*.{cs,vb}]

# Задание важности правилу
dotnet_diagnostic.CA1720.severity = warning

# Ограничение области проверки по модификатору
dotnet_code_quality.CA1720.api_surface = private, internal
```

Анализаторы StyleCop

Назначение

StyleCopAnalyzers (ссылка на репозиторий)— утилита, которая позволяет анализировать код на наличие проблем с качеством и стилем кода. Если анализатор обнаруживает нарушения правил, появляется сообщение с префиксом "SA" в виде предложения, предупреждения или ошибки — в зависимости от того, как настроено каждое правило.

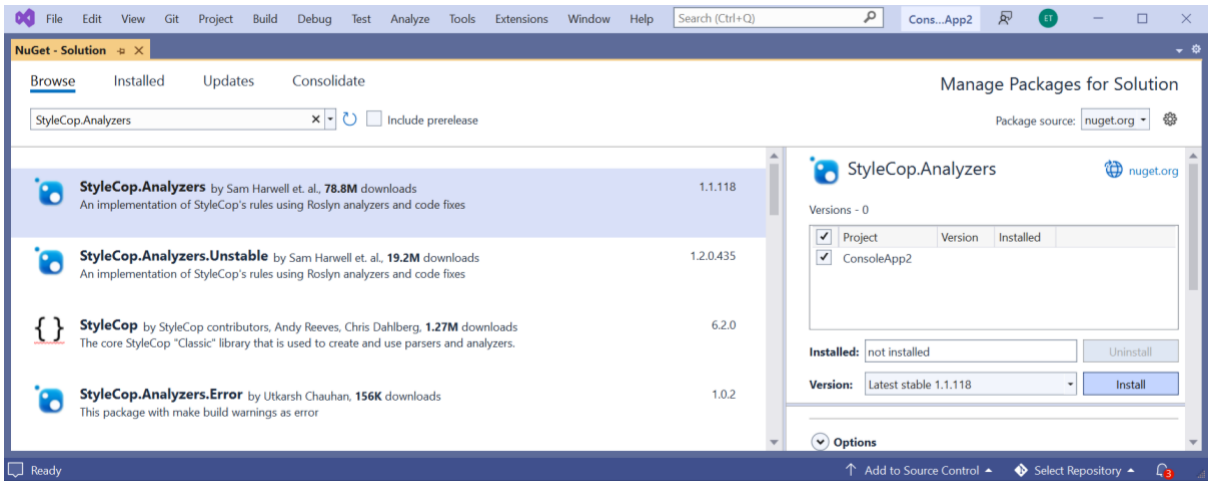
Установка

StyleCopAnalyzers устанавливается в проект с помощью пакетного менеджера NuGet.

Чтобы установить утилиту, необходимо открыть консоль NuGet (в Visual Studio: Tools → NuGet Package Manager → Package Manager Console) и выполнить команду:

```
Install-Package StyleCop.Analyzers
```

Также можно выполнить установку с помощью графической оболочки NuGet. Для этого необходимо в Visual Studio перейти: Tools → NuGet Package Manager → Manage Packages for Solution, во вкладке Browse выполнить поиск StyleCop.Analyzers и установить утилиту для проекта:



Настройка правил

Конфигурация StyleCop.Analyzers выполняется с помощью двух отдельных файлов.

Файл `stylecop.ruleset`

Файлы `*.ruleset` — это XML-файлы, которые используются в Visual Studio для настройки встроенных анализаторов кода и могут быть также использованы для настройки StyleCop.Analyzers.

Данный файл позволяет включать / отключать правила и настраивать уровень их важности. Структура этих файлов описана в документации Visual Studio и в случае использования StyleCop.Analyzers выглядит следующим образом:

```
<?xml version="1.0"?>
<RuleSet Name="StyleCop.Analyzers rules"
         Description="StyleCop.Analyzers rules description"
         ToolsVersion="14.0">
    <Rules AnalyzerId="StyleCop.Analyzers"
           RuleNamespace="StyleCop.Analyzers.SpecialRules">
        <Rule Id="SA0001" Action="Warning" />
        <Rule Id="SA0002" Action="Warning" />
    </Rules>
    <Rules AnalyzerId="StyleCop.Analyzers"
           RuleNamespace="StyleCop.Analyzers.SpacingRules">
        <Rule Id="SA1000" Action="Warning" />
        <Rule Id="SA1001" Action="Warning" />
    </Rules>
</RuleSet>
```

Конфигурация каждого правила состоит из номера правила `Rule Id` и уровня его важности — `Action`. Все правила сгруппированы по категориям `RuleNamespace` и анализаторам `AnalyzerId`. В данном случае `AnalyzerId` всегда равен `"StyleCop.Analyzers"`, а категории `RuleNamespace` могут быть следующие:

- `SpecialRules` — правила для контроля ошибок конфигурации
- `SpacingRules` — правила расстановки пробелов около операторов
- `ReadabilityRules` — правила для контроля читаемости кода
- `OrderingRules` — правила для контроля порядка размещения кода в файле
- `NamingRules` — правила именования элементов
- `MaintainabilityRules` — правила для облегчения сопровождения кода
- `LayoutRules` — правила, определяющие оформление кода
- `DocumentationRules` — правила для контроля документации к коду
- `AlternativeRules` — правила для нестандартной настройки стиля кода

Описание правил и их `Rule Id` перечислены в документации по категориям. Уровни важности `Action` могут быть следующие:

- `Error` — нарушение правила приводит к ошибке при сборке проекта
- `Warning` — нарушение правила приводит к предупреждению при сборке проекта
- `Info` — при нарушении правила появляются информационные сообщения
- `Hidden` — нарушение правил будет невидимо для пользователя (отображается только в меню рефакторинга)

- `None` — отключение правила

Чтобы использовать в Visual Studio правила, определенные в `*.ruleset` файле, необходимо в конфигурации проекта (т. е. в файле `*.csproj`) задать к нему путь. Например, если настройки StyleCop.Analyzers определены в `stylecop.ruleset`, тогда конфигурация проекта будет выглядеть следующим образом:

```
<PropertyGroup>
  . . .
  <CodeAnalysisRuleSet>stylecop.ruleset</CodeAnalysisRuleSet>
</PropertyGroup>
```

Файл `stylecop.json`

Данный файл позволяет указывать некоторые параметры проекта (название компании, информацию об авторских правах и т. д.), а также конфигурировать отдельные правила. Файл `stylecop.json` имеет следующую базовую структуру:

```
{
  "$schema": "https://raw.githubusercontent.com/DotNetAnalyzers/…",
  "settings": {
    "documentationRules": {
      "companyName": "PlaceholderCompany"
    }
  }
}
```

В `"$schema"` задается ссылка на описание json-схемы файла, что позволяет использовать возможности IntelliSense при редактировании файла в Visual Studio (автодополнение, краткая информация и т. д.). Далее в `"settings"` перечисляются необходимые настройки. Перечень всех возможных настроек можно найти в документации. Пример настройки отступов:

```
{
  "$schema": "https://raw.githubusercontent.com/DotNetAnalyzers/…",
  "settings": {
    "indentation": {
      "indentationSize": "4",
      "tabSize": "4",
      "useTabs": "false"
    }
  }
}
```

Чтобы использовать в Visual Studio правила, определенные в `stylecop.json`, необходимо в конфигурации проекта (т. е. в файле `*.csproj`) задать

```
<PropertyGroup>
```

```
  . . .
</PropertyGroup>
<ItemGroup>
  <AdditionalFiles Include="stylecop.json" />
</ItemGroup>
```

Перечень неучтенных правил

1. Члены перечислений должны быть в стиле паскаль.
2. Длина строк --- 78 символов.
3. Комментарии должны быть над строкой с кодом, а не справа.
4. Слова в названиях непубличных элементов не должны разделяться знаком подчеркивания.
5. Объявления полей класса должны быть на отдельных строках.