Санкт-Петербургский государственный университет

Глазырин Кирилл Максимович

Выпускная квалификационная работа

Поддержка языка T4 в среде разработки Rider

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2017 «Программная инженерия»*

Научный руководитель:
к. т. н., доцент Литвинов Ю. В.

Консультант:
старший разработчик ООО «Интеллиджей Лабс» Кирсанов А. Ю.

Рецензент:
разработчик ООО «Интеллиджей Лабс» Аудучинок Е. П.

Санкт-Петербург
2021

Saint Petersburg State University

Kirill Glazyrin

Bachelor's Thesis

T4 language support in Rider IDE

Education level: bachelor

Speciality *09.03.04 "Software Engineering"*

Programme *CB.5080.2017 «Software Engineering»*

Scientific supervisor:
C. Sc., docent Yurii Litvinov

Consultant:
Senior Software Developer at "JetBrains" Alexander Kirsanov

Reviewer:
Software Developer at "JetBrains" Eugene Auduchinok

Saint Petersburg
2021

Оглавление

Введение

При разработке программного обеспечения может возникать потребность написания простого и несодержательного кода или создания большого количества однотипных данных; автоматизация этого процесса может значительно увеличить производительность труда программиста. Один из способов автоматизировать это — написать программу, которая генерирует код или данные для проекта.

Поэтому существуют инструменты, упрощающие написание программ для кодогенерации. Один из таких инструментов — T4 [7]. T4 — язык программирования, созданный компанией Microsoft. Он содержит фрагменты кода на C#, которые исполняются для того, чтобы сгенерировать текст или код. Благодаря своему удобству этот инструмент широко используется: на платформе хостинга кода GitHub по запросу

```
``template OR parameter OR output OR assembly OR import OR
  include extension:tt extension:t4 extension:ttinclude''
```

на момент написания данной работы находится более 570'000 файлов[1].

У этого инструмента был важный недостаток: ни одна среда разработки не предоставляла полноценную поддержку этого языка «из коробки». Единственный инструмент, предоставлявший полноценную возможность исполнять такие файлы — Visual Studio, среда разработки от компании Microsoft. Но и она не предоставляла никакой интеллектуальной поддержки при редактировании этих файлов.

Более того, Visual Studio была единственной средой разработки, предоставлявшей полноценную поддержку исполнения файлов на этом языке. Из-за этого данная технология была привязана к Visual Studio, и все, кто работал с проектами с T4, оказывались привязаны к ней и к операционной системе Windows.

В частности, они не могли использовать Rider [3] — кроссплатфор-

[1]В данном запросе фигурирует ограничение на содержание файлов: файлы, удовлетворяющие этому запросу, содержат хотя бы одно из указанных слов: template, parameter, output и так далее. Это означает, что фактическое количество файлов на языке T4 ещё больше. Это ограничение было добавлено, потому что GitHub не допускает запросы, не накладывающие ограничения на содержимое файлов. Данные слова были выбраны, потому что они является частью ключевых конструкций языка, поэтому можно считать, что почти все файлы на этом языке содержат хотя бы одно из них.

менную среду разработки от компании JetBrains.

Поэтому возникла идея поддержать язык T4 в Rider. Для того, чтобы проблему можно было считать решённой, в среде разработки Rider должна появиться возможность исполнять файлы на языке T4 и интеллектуальная поддержка при их редактировании, которая включает в себя, но не ограничивается такими функциями, как подсветка синтаксиса, автодополнение, анализ кода на потенциальные проблемы, рефакторинги, показ документации.

Для того, чтобы у языка T4 был полноценный кроссплатформенный способ исполнения и кроссплатформенный инструмент, предоставляющий полноценную поддержку сразу после установки, было принято решение создать плагин к Rider, который добавляет необходимую функциональность. Для упрощения задачи было принято решение не писать новый плагин, а адаптировать существующий плагин к ReSharper[2] к запуску в Rider, переиспользовать код из него и расширить его функциональность до необходимого уровня.

[2]Resharper — расширение к Visual Studio, повышающее производительность труда разработчиков [4].

1. Постановка задачи

Целью данной работы является создание плагина к среде разработки Rider, добавляющего поддержку языка T4. Для её выполнения были поставлены следующие задачи:

1. Адаптировать существующий плагин к ReSharper к запуску в Rider.

2. Добавить в плагин возможность исполнять файлы и отлаживать их исполнение.

3. Улучшить интеллектуальную поддержку редактирования файлов, предоставляемую плагином.

4. Протестировать на типичных примерах файлов, автоматизировать это тестирование.

5. Получить обратную связь от пользователей и исправить возможные недочёты.

2. Обзор предметной области

2.1. Синтаксис T4

Файлы на языке T4 состоят из следующих элементов:

- директив, которые регулируют то, как эти файлы исполняются. В частности, они позволяют настраивать то, какой язык будет использоваться в других фрагментах файла; какое расширение будет у файла, создающегося при исполнении T4-файла; ссылки на какие сборки должны присутствовать при компиляции файла; и многое другое. Полный список возможных директив можно найти на сайте документации Microsoft [9];

- блоков текста, который попадает в результат генерации напрямую;

- блоков кода на языке C# или Visual Basic .NET, которые комбинируют блоки текста. Например, при помощи них можно вставлять текст в результат генерации несколько раз или вставлять/не вставлять его в зависимости от некоторых условий. Кроме того, в особом варианте таких блоков могут быть объявлены функции, которые можно использовать в других блоках кода.

2.1.1. Директива include

Одна из наиболее важных директив, используемых в T4 — include. Она позволяет дословно вставлять содержимое одного файла в другой. Это может быть использовано для того, чтобы переиспользовать код на языке T4: один и тот же файл может быть включён в несколько других файлов. Блоки кода в файле с общим кодом могут не быть корректным с точки зрения языка, на котором они написаны. Например, они могут содержать вызовы функций, не объявленных в этом файле. Для того, чтобы быть корректными, файлы, включающие в себя такие неполные файлы, должны содержать объявление символов, используемых, но не объявленных в инклюдах, или включать в себя файлы с объявлением

этих символов. Для того, чтобы файлы с общим кодом не исполнялись, можно сменить их расширение с расширения по умолчанию (*.tt) на другое. Язык T4 позволяет включать файлы с любым расширением, но Microsoft рекомендует использовать расширение ttinclude [8].

2.1.2. Виды T4-файлов

Файлы на языке T4 бывают трёх видов: исполняемые, препроцессящиеся и файлы, включаемые в другие. Первые два вида имеют одно и то же расширение (tt) и отличаются только метаданными, записанными в проектный файл. Исполняемые файлы можно исполнить, то есть применить все директивы, находящиеся в них, исполнить блоки кода и получить некоторый результат, который становится частью проекта. Препроцессящиеся фалы используются в том случае, если необходимо сгенерировать некоторый результат не во время написания кода, а во время исполнения программы: в результате препроцессинга создаётся файл, содержащий код, при исполнении которого получается тот же результат, что получился бы при исполнении исходного файла. Файлы, включаемые в другие, не имеют собственной кодогенерации. То, как используется код, написанный в них, зависит от того, в какие файлы они включены.

2.1.3. Макросы

Многие директивы могут содержать пути. Например, это могут быть пути до сборок в директиве assembly или пути до файлов *.ttinclude в директиве include. Эти пути могут быть как относительными (начиная с директории, в которой находится файл), так и абсолютными. Для того, чтобы использовать абсолютные пути, но при этом не завязываться на конкретную структуру директорий, в путях можно использовать макросы — переменные, которые раскрываются в некоторое значение при компиляции файла. Например, есть макросы, раскрывающиеся в путь до папки решения и путь до папки проекта.

2.2. Сложности языка T4

Язык T4 имеет ряд особенностей, которые делают его поддержку сложнее:

- многие маросы, которые можно использовать в директивах, содержат некоторую информацию о структуре решения. Эта информация доступна при работе среды разработки и при сборке решения. Инструменты командной строки не имеют доступа к ней, потому что её получение требует загрузки решения, что включает в себя разбор всех его проектных файлов, что было бы недопустимо с точки зрения производительности. Поэтому полноценная поддержка исполнения T4 должна быть реализована как плагин к среде разработки или расширение к инструменту сборки;

- особую сложность представляет тот факт, что файлы *.ttinclude могут содержать код, который не корректен вне файлов, которые включают его. В отличие от языка C++, в котором принято создавать заголовочные файлы с объявлениями символов, и включать их в места использования этих символов, в T4 нет такого понятия, как “заголовочный файл”. Это означает, что в T4 нет простого способа по использованию символа определить, где он объявлен. Он может быть объявлен не только в файле с его использованием или в любом из файлов, которые он включает. Он может быть объявлен и в любом из файлов, в которые включается файл с использованием символа. Более того, он может быть объявлен в любом файле, который включён в файл, в который включён файл с использованием символа. Более того, все упомянутые включения могут быть как прямыми, так и с некоторым количеством посредников, включённых в предыдущий файл и включающих следующий. Кроме того, один и тот же символ может иметь различную семантику в различных контекстах.

2.3. Существующие решения

Необходимость исполнять T4-файлы вне Visual Studio имеет возможные решения:

- среда разработки Visual Studio предоставляет инструменты командной строки, позволяющие исполнять эти файлы. Однако они поставляются с Visual Studio, что накладывает некоторые ограничения с точки зрения лицензионной политики;
- существует кроссплатформенная реализация генератора T4 с открытым кодом [10].

Однако оба этих решения имеют ограниченную применимость, потому что они не позволяют исполнять файлы, содержащие макросы.

Проблема отсутствия интеллектуальной поддержки при редактировании T4-файлов тоже имеет ряд возможных решений:

- можно установить расширение к Visual Studio, добавляющее поддержку этого языка. Кроме того, можно установить в Visual Studio расширение ReSharper и установить в него плагин, добавляющий поддержку этого языка. Однако это решение не кроссплатформенное: оно опирается на Visual Studio и операционную систему Windows;
- можно установить расширение к редактору, например, Visual Studio Code. Однако интеллектуальная поддержка языка C# в блоках кода внутри T4 гораздо слабее поддержки C# в полноценной IDE. В частности, все плагины, добавляющие поддержку T4 в Visual Studio Code, ограничиваются подсветкой синтаксиса.

2.4. ReSharper

ReSharper — расширение к среде разработки Visual Studio. Существует возможность писать плагины к нему; существует документация о том, как делать это [2].

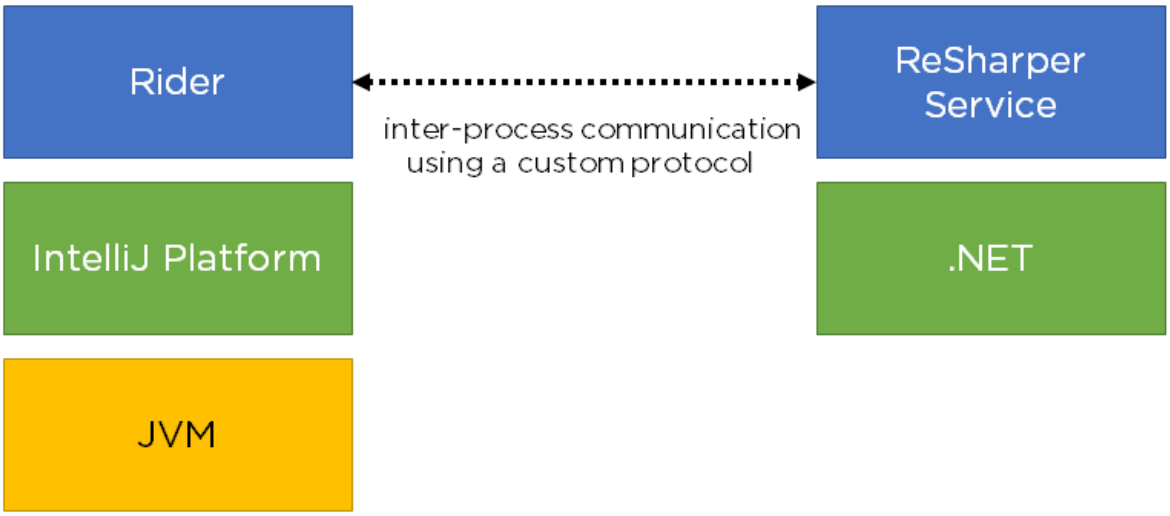

Рис. 1: Архитектура среды разработки Rider [1].

2.5. Rider

Rider [3] — кроссплатформенная среда разработки для .NET от компании JetBrains. Её архитектура изображена на рис. 1. Данная среда разработки использует два основных процесса: процесс на JVM, который отрисовывает пользовательский интерфейс, и процесс на .NET, исполняющий содержательную логику анализа файлов. Процесс на .NET исполняет тот же код, что и ReSharper; таким образом избегается дублирование кода интеллектуальной поддержки между двумя продуктами.

2.6. Архитектура плагинов к Rider

Плагины к Rider отражают архитектуру этой среды разработки и состоят из части, исполняющейся на JVM, и части, исполняющейся на .NET. При этом часть, исполняющаяся на .NET, имеет очень много общего с плагином к ReSharper, благодаря чему переиспользование кода возможно не только между ReSharper и бэкендом Rider, но и между плагинами к ReSharper и бэкендами плагинов к Rider.

3. Предлагаемое решение

В ходе данной работы был реализован плагин к среде разработки Rider. Плагин называется ForTea. Его исходный код был выложен на сервис GitHub по адресу https://github.com/JetBrains/ForTea. В настоящий момент плагин к Rider является “забандленным”, то есть собирается и поставляется вместе с продуктом. Из того же самого исходного кода можно собрать плагин к ReSharper. Плагин к ReSharper не поставляется вместе с ним, но он доступен к установке через встроенный в ReSharper инструмент Extension Manager, а также через сайт плагинов по ссылке https://plugins.jetbrains.com/plugin/13469-fortea.

3.1. Основа плагина

Так как бэкенд Rider почти полностью основан на ReSharper, из одного и того же кода можно собирать как плагин к Rider, так и плагин к ReSharper [5]. При этом плагин к Rider и плагин к ReSharper могут разделять значительную часть общего кода. Поэтому было принято решение не писать плагин с нуля, а взять за основу уже существовавший плагин к ReSharper, называющийся ForTea [6], и адаптировать его к запуску в Rider. Однако в ходе работ код плагина был почти полностью переписан. Для того, чтобы подтвердить это, был создан инструмент, вызывающий git blame для каждого файла в проекте и агрегирующий информацию о количестве строк, принадлежащих каждому пользователю, делавшему коммит в проект. Согласно результатам его исполнения в репозитории проекта, в настоящий момент исходному автору плагина принадлежат менее чем для 3% строк кода. Исходный код упомянутого инструмента был выложен на сервис GitHub по адресу https://github.com/kirillgla/GitBlameAggregator.

3.2. Архитектура плагина

Плагины к Rider во многом отражают архитектуру этой среды разработки и состоят из двух основных частей: фронтендной части плаги-

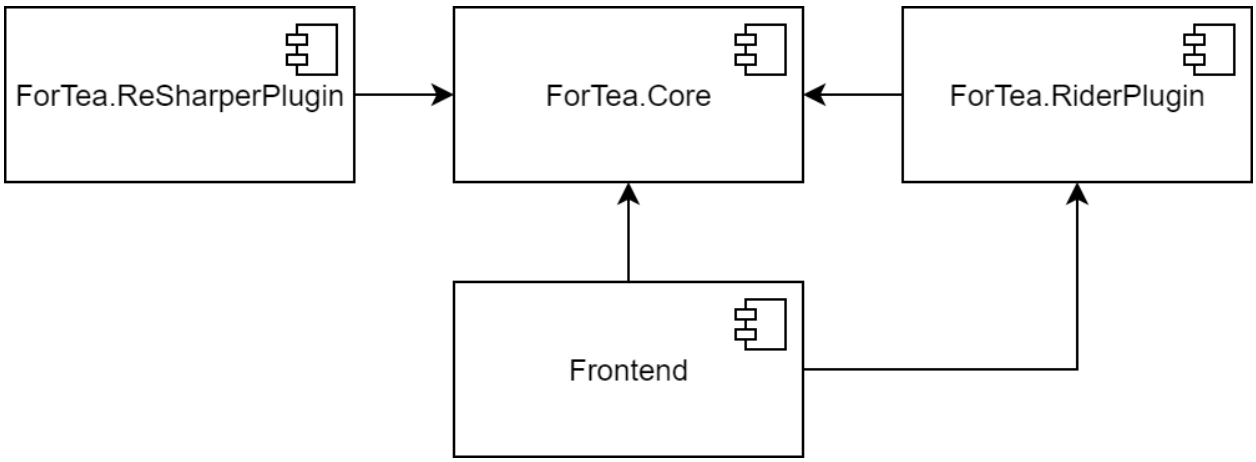

Рис. 2: Ключевые архитектурные элементы кода.

на, которая загружается в процесс фронтенда Rider, и бэкендной части плагина, которая загружается в процесс бэкенда Rider. Необходимость собирать два плагина из одного кода также во многом диктует их архитектуру. Основными элементами архитектуры являются:

- ForTea.Core — сборка, содержащая код, общий для плагина к Rider и плагина к ReSharper. Она используется в обоих плагинах, и к ней относится большая часть кода проекта;
- ForTea.ReSharperPlugin — сборка, содержащая код, относящийся только к плагину к ReSharper. В частности, она содержит логику взаимодействия с Visual Studio;
- ForTea.RiderPlugin — сборка, содержащая код, относящийся только к плагину к Rider. В частности, она содержит логику взаимодействия с фронтендной частью плагина;
- Frontend — фронтендная часть плагина к Rider. Она содержит логику подсветки синтаксиса и пользовательского интерфейса. Эта часть делегирует большую часть нетривиальной функциональности в ForTea.RiderPlugin.

Связи между данными элементами изображены на рис. 2.

3.3. Обзор функциональности плагина

Значительная часть поддержки редактирования уже была в плагине до его адаптации. Она включала в себя такие элементы, как поддержку редактирования C# в блоках кода. Поддержка редактирования блоков кода на C# делегируется в среду разработки, и там присутствует вся необходимая функциональность, поэтому эта часть не нуждалась в расширении. Однако поддержка редактирования других элементов синтаксиса могла быть улучшена, что и было сделано.

3.3.1. Поддержка редактирования

Поддержка редактирования элементов синтаксиса T4 была расширена более качественной поддержкой макросов: в них был улучшен отчёт об ошибках и добавлены подсказки, показывающие во что раскроется макрос при исполнении.

3.3.2. Подсветка синтаксиса в блоках текста

Так как T4 позволяет задавать расширение фалов, которые генерируются в ходе исполнения, при помощи этого языка можно генерировать код на разных языках. Так как расширение выходного файла явно указывается в коде на T4, выходной язык можно определить во время анализа файла. Это позволяет предоставить подсветку синтаксиса в блоках текста. Для того, чтобы подсветить синтаксис в блоках текста, плагин при помощи специального лексера выделяет все регионы файла, содержащие только код на C#, совмещает их в единый блок текста, затем на основе расширения выходного файла выбирает и запускает лексер для подсветки синтаксиса на этом блоке. Получающийся текст, как правило, не является корректным кодом на C#, но он является достаточным для того, чтобы подсветить основные элементы синтаксиса, например, ключевые слова и строковые литералы. Пример результатов работы такой подсветки синтаксиса можно видеть на рис. 3.

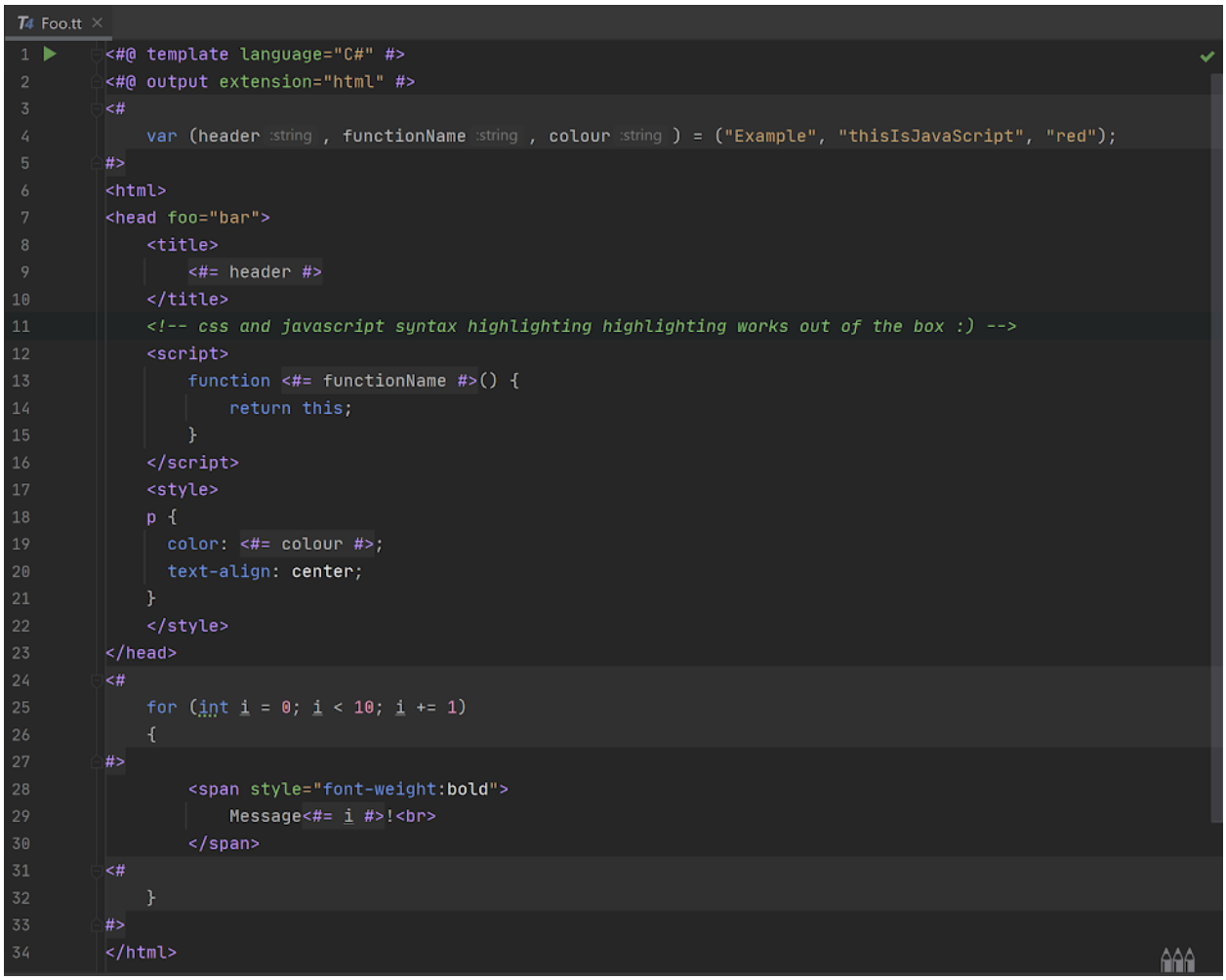

Рис. 3: Пример подсвеки синтаксиса в блоках текста.

3.3.3. Исполнение файлов

Плагин к Rider позволяет исполнять файлы на языке T4. Для того, чтобы исполнить пользовательский код на языке T4, среда разработки транслирует его в C#, затем компилирует получившийся код при помощи компилятора Roslyn и исполняет полученный бинарный файл.

3.3.4. Отладка файлов

Кроме исполнения, плагин позволят отлаживать файлы. При генерации кода на C# плагин вставляет директивы `#line`, которые позволяют составить соответствие между строчками в сгенерированном коде и строчками в исходном файле. Затем компилятор, ориентируясь на эти директивы, при кодогенерации создаёт отладочную информацию, которая ссылается на T4-файл как на файл с исходным кодом. Затем отладчик среды разработки, опираясь на эти метаданные, предоставляет все те же элементы функциональности, что и для файлов на C#. Таким образом, при отладке T4 на самом деле происходит отладка кода на .NET. Благодаря этому возможен прозрачный переход между отладкой T4-файлов и библиотек: с точки зрения отладчика файлы на T4 ничем не отличаются от обычных файлов на C#.

3.3.5. Препроцессинг файлов

Язык T4 подразумевает возможность не исполнять файлы сразу, а генерировать по ним код на C#, который добавляется в проект. При исполнении этого кода создаётся тот же результат, что получился бы при исполнении файла. Эта функция тоже поддержана в плагине. При этом переиспользуется значительная часть кода кодогенерации перед исполнением.

3.3.6. Автоматическое исполнение и препроцессинг

Для того, чтобы упростить работу с файлами на языке T4, были добавлены автоматические исполнение и препроцессинг файлов: при

сохранении файлов на диск производится соответствующая операция. Это позволяет исключить из цикла работы с файлом дополнительный шаг. При этом исполнение и препроцессинг происходят в отдельном процессе, поэтому они не прерывают пользовательское взаимодействие с редактором.

3.4. Обеспечение качества

Как бэкенд, так и фронтенд плагина покрыты тестами. Тесты бэкенда покрывают функциональность, полностью реализованную на бэкенде: typing assistance, выделение парных скобок, окно структуры кода, форматирование и многое другое. Тесты фронтенда покрывают функциональность, требующую наличия обеих частей плагина или полностью реализованную на фронтенде: исполнение и препроцессинг файлов, подсветку синтаксиса и многое другое.

3.5. Обратная связь

Так как плагин является частью продукта Rider, пользователи, имеют возможность предоставлять обратную связь посредством issue-трекера продукта. Внешние пользователи завели в трекере 44 issue, связанных с плагином, 30 наиболее критичных из которых были исправлены.

3.6. Документация

Сборка, исполнение и отладка обоих плагинов, а также запуск и отладка фронтендных и бэкенных тестов задокументированы. Инструкции находятся по ссылке https://github.com/JetBrains/ForTea/tree/master/Documentation.

4. Обзор ключевых алгоритмов

В ходе работы над плагином был разработан ряд важных алгоритмов, необходимость которых обоснована особенностями языка T4:

- построение и поддержка в актуальном состоянии графа включения файлов;
- синтаксический анализ файла вместе с его явными и неявными зависимостями в графе инклюдов;
- оптимизация синтаксического анализа за счёт копирования уже построенных деревьев.

4.1. Граф включения файлов

Имея доступ только к исходному коду файла, можно определить, какие файлы он включает в себя. Однако бывает нужно определять, какие файлы включают его. Для того, чтобы удовлетворить эту необходимость, в плагине реализована таблица, содержащая информацию о включении файлов. При первом открытии проекта эта таблица строится на основе всех файлов проекта; при каждой модификации файла данные о включении, связанные с этим файлом, обновляются. Кроме того, таблица сохраняется на диск средствами среды разработки, что позволяет не строить её при повторных открытиях проекта.

4.2. Контекст синтаксического анализа

Из-за существования директивы include нарушается ряд инвариантов, на которые опирается код поддержки C#. Например, инвариант о том, что локальные переменные используются только в рамках одного файла. Поэтому для предоставления полноценной поддержки редактирования необходимо отслеживать контекст, в котором находится включаемый файл, то есть иметь список файлов, которые могут содержать объявление символов, используемых в текущем. Это реализовано

через комбинацию двух поисков в ширину по графу включений. Первый поиск находит ближайший файл, который включает текущий и при этом не включается никуда, а второй — все файлы, включённые в этот корень. Это необходимый и достаточный контекст. Он необходим, потому что поиск нельзя остановить раньше: любой из промежуточных файлов не обязан содержать корректный код, потому что этот файл, скорее всего, не исполняют. При этом он достаточный, потому что файлы, не включённые в другие, должны быть корректными, потому что файл, не включаемый никуда, скорее всего, исполняют, потому что у него не может быть других предназначений, поэтому контекст, основанный на таком файле, должен содержать определения всех используемых символов. Добавление такого контекста позволило сделать плагин применимым в значительно более крупном количестве случаев. Ранее существовавший плагин для символов, присутствующих только в контексте, показывал ошибки.

4.3. Оптимизация синтаксического анализа

Необходимость разбирать контекст вместе с самим файлом делает синтаксический анализ более трудоёмкой операцией. При этом часто бывает так, что для целой группы файлов контекст один и тот же, и их синтаксические деревья совпадают с точностью до незначительных деталей. Поэтому для того, чтобы получить абстрактное синтаксическое дерево для файла, часто бывает не нужно его строить; бывает достаточно просто запустить один обход в глубину, по уже построенному дереву, который копирует все узлы, которые встречает. Плагин использует этот факт и строит деревья только для корневых файлов. Для файлов, включённых в другие, клонируются деревья корневых файлов. При этом разбор и копирование синтаксических деревьев заточены под работу в среде разработки: когда происходит печать в редакторе, плагин не дожидается, пока построится дерево, которое он будет клонировать, чтобы обновить подсветку. Вместо этого он клонирует те части дерева, которые точно не изменились, и самостоятельно разбирает

остальные.

Рис. 4: Распределения времён исполнения операций.

5. Эксперимент

Для того, чтобы оценить эффективность проделанной оптимизации синтаксического анализа, был проведён эксперимент: было замерено время разбора крупного файла обычным парсером и время разбора того же самого файла клонирующим парсером.

5.1. Условия эксперимента

Измерения проводились на машине с процессором Intel(R) Core(TM) i7-8750H CPU @2.20GHz. В качестве входных данных для алгоритмов служил набор сгенерированных файлов, которые содержали все основные элементы синтаксиса языка T4. Код, производящий измерения, выложен в репозиторий плагина.

5.2. Результаты

Результаты измерений приведены на рис. 4 и в таблице 1.

Операция	E, мс	σ, мс
Разбор	18.2	1.9
Клонирование	9.5	1.1

Таблица 1: Статистические данные времён разбора и исполнения. E — математическое ожидание. σ — стандартное отклонение.

Эти данные показывают, что клонирование деревьев производится почти в два раза быстрее, чем их построение.

6. Ограничения

Поддержка языка T4, предоставляемая созданным плагином, не полностью соответствует описанию на сайте Microsoft. В частности, присутствуют следующие ограничения:

- отсутствует возможность создания собственных процессоров директив и, как следствие, возможность использования собственных директив;
- Плагины к Visual Studio могут расширять список директорий, в которых компилятор T4 ищет файлы для включения. Аналогичную функциональность возможно реализовать и в плагине к Rider, но гораздо сложнее.

7. Заключение

В ходе данной работы был создан плагин к среде разработки Rider, добавляющий поддержку языка T4. Для этого было сделано следующее:

1. Существующий плагин к ReSharper был адаптирован к запуску в Rider. Исходный код нового плагина был выложен на сервис GitHub по адресу https://github.com/JetBrains/ForTea.

2. В плагин была добавлена возможность исполнять файлы и отлаживать их исполнение.

3. Интеллектуальная поддержка редактирования файлов, предоставляемая плагином, была расширена.

4. Плагин был протестирован на типичных примерах файлов; это тестирование было автоматизировано.

5. Была получена обратная связь от пользователей. Наиболее критичные недочёты были исправлены.

Список литературы

[1] Balliauw Maarten. Rider front end plugin development // blog. — 2017. — URL: https://blog.jetbrains.com/dotnet/2017/02/07/rider-front-end-plugin-development/ (дата обращения: 10.25.2020).

[2] JetBrains. ReSharper DevGuide. — 2017. — URL: https://www.jetbrains.com/help/resharper/sdk/README.html (online; accessed: 17).

[3] JetBrains. Rider // overview. — 2019. — URL: https://www.jetbrains.com/rider/ (дата обращения: 28.10.2019).

[4] JetBrains. ReSharper // overview. — 2021. — URL: https://www.jetbrains.com/resharper/ (дата обращения: 02.05.2021).

[5] Koch Matthias. Writing plugins for ReSharper and Rider // blog. — 2019. — URL: https://blog.jetbrains.com/dotnet/2019/02/14/writing-plugins-resharper-rider/ (дата обращения: 10.25.2020).

[6] Lebosquain Julien. ForTea // GitHub. — 2013. — URL: https://github.com/MrJul/ForTea/ (дата обращения: 16.25.2020).

[7] Microsoft. Code Generation and T4 Templates. — 2016. — URL: https://docs.microsoft.com/en-us/visualstudio/modeling/code-generation-and-t4-text-templates (дата обращения: 02.05.2021).

[8] Microsoft. Guidelines for Writing T4 Text Templates. — 2016. — URL: https://docs.microsoft.com/en-us/visualstudio/modeling/guidelines-for-writing-t4-text-templates (дата обращения: 17.12.2020).

[9] Microsoft. T4 Text Template Directives. — 2016. — URL: https://docs.microsoft.com/en-us/visualstudio/modeling/t4-text-template-directives (дата обращения: 16.12.2020).

[10] Mono. Mono.TextTemplating. — 2020. — URL: https://github.com/mono/t4 (online; accessed: 18).