

Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра системного программирования

Обеспечение работы и поддержка консистентности
модели семантической подсветки кода в IDE
JetBrains Rider

Курсовая студента 1-го курса магистратуры, 546 группы
Кирсанова Александра Юрьевича

Научный руководитель: профессор Терехов А.Н.
Рецензент: руководитель отдела .NET разработки
JetBrains, Сергей Шкредов

Введение

Семантическая подсветка является неотъемлемой частью функциональности, предлагаемой современными текстовыми редакторами и IDE. Подсветка ключевых слов и различных идентификаторов позволяет разработчику быстрее ориентироваться в коде программы. Кроме того, markup-модель (модель разметки) выступает в роли одного из основных способов взаимодействия с пользователем и помогает сообщить программисту об ошибках и возможных недочетах.

ReSharper предоставляет полноценную поддержку для более чем десяти языков, частью которой является и семантическая подсветка кода. Более тысячи различных инспекций ReSharper'a позволяют выявить ошибки компиляции, времени выполнения, а также избыточные и неоптимальные конструкции, каждая из которых будет подсвечена в редакторе.

Постановка задачи

В ходе данной работы необходимо обеспечить работу семантической подсветки кода в проекте Rider, новой IDE на базе IntelliJ IDEA и ReSharper. В рамках курсовой работы предполагается создать межязыковой механизм передачи highlighter'ов (элементов модели подсветки кода) из ядра ReSharper в редактор IDEA, сохраняющий корректное поведение и поддерживающий консистентность такой модели при конкурентных модификациях кода.

Ход работы

Для удобства читателя весь ход работы разделён на две части. В первой из которых обсуждается представление маркап-модели, выбор уровня для передачи данных и реализация механизма отправки маркап-модели, а во второй – сложности, возникающие в условиях постоянных изменений документов и соответствующих маркап-моделей, а также оптимизации, необходимые для обеспечения работы такой системы.

Архитектура JetBrains Rider

Rider – новый проект компании JetBrains, это кросс-платформенная интегрированная среда разработки, сочетающая в себе возможности ReSharper и IntelliJ IDEA. Последняя выступает для Rider в качестве front-end'а, предоставляя пользовательский интерфейс, работу с VCS, и другую функциональность, не связанную со статическим анализом и проектной моделью. ReSharper запущен в отдельном процессе, как консольное приложение, и выступает в качестве back-end'а, сервера, на котором происходит построение проектной модели, производятся все анализы и который предоставляет сервисы для манипуляций над кодом (рефакторинг, быстрые исправления ошибок, автоматическое дополнение кода, форматирование и т.д.) и навигации (поиск использования, переход к декларации, поиск классов, файлов и методов и т.д.). Коммуникация между двумя процессами осуществляется с помощью специального бинарного протокола, созданного в команде Rider. Для каждого из сервисов, предоставляемых ReSharper-хостом, выбран уровень представления данных, позволяющий сократить их объем при передаче, к рассмотрению конкретных примеров взаимодействия мы вернёмся дальше в этой главе. Пользователь выполняет какое-нибудь действие (вызов быстрого исправления, нажатие комбинации клавиш, или клик на специальные объекты UI), управление передаётся в переопределенную компоненту, которая передаёт на back-end необходимые данные и обрабатывает полученный результат, чаще всего снова интегрируясь в IDEA.

Всё взаимодействие между ReSharperHost'ом и IDEA (за небольшим исключением) является асинхронным. Так как все сложные операции происходят в другом процессе от редактора, в котором работает пользователь, а во front-end данные приходят в готовом виде и в небольшом количестве, маловероятны ситуации зависания пользовательского интерфейса и увеличение времени отклика на пользовательские изменения.

Выбор подходящего уровня передачи данных

Семантическая подсветка и результаты инспекций, которые видны пользователю в виде подсвеченного текста в исходном коде, являются результатом сложных статических анализов, выполняемых демоном при первом открытии и повторяющихся на каждое изменение в файле. Для обеспечения работы всей этой системы необходимо глубокое знание о коде, а для передачи этой информации (синтаксических деревьев и других структур, связанных с PSI) на сторону IDEA понадобятся слишком большие объемы памяти, а учитывая скорость и масштабы изменения таких структур, не получится добиться хорошей производительности.

В самом деле для отображения результатов работы ReSharperHost'a в редакторе нужно гораздо меньше данных: отрезка текста; эффектов, которые нужно применить к тексту (например, подчеркнуть его красной волной, если это синтаксическая ошибка).

Таким образом, в качестве передаваемых данных предполагается использовать минимальное количество информации, необходимой для ОТОБРАЖЕНИЯ элементов подсветки.

Элемент маркап модели (Highlighter)

Хайлайтер (Highlighter) – объект, представляющий подсвеченный фрагмент текста. Примерами подсвеченного текста могут служить ключевые слова, строки, комментарии, декларации и вызовы методов, имена переменных и друг идентификаторов, ошибки, предупреждения и

недостижимые участки кода. Для каждого из приведенных элементов AST и результатов инспекций кода, существует свой хайлайтер, который накладывается на текст исходного документа.

Хайлайтер является объектом уровня представления, это результат работы анализаторов, который содержит все необходимые данные для отрисовки такого эффекта. В следствие этого в нём содержится большое количество информации

- 1) абсолютные смещения начала и конца отрезка в документе, который необходимо подсветить
- 2) строка с документацией или сообщением об причине ошибки, который появится, если пользователь наведёт мышкой на подсвеченный текст
- 3) уровень опасности такого хайлайтера (information, hint, suggestion, warning, error)
- 4) тип текстового эффекта
 - a) волна снизу
 - b) нижнее подчеркивание
 - c) пунктирная линия
 - d) выделение заднего фона текста
 - e) ...
- 5) цвет эффекта
- 6) тип шрифта, его толщина
- 7) ...

Поскольку маркап модель постоянно претерпевает изменения в своей структуре и может провоцировать массивные удаление и добавление новых хайлайтеров, накладные ресурсы от сериализации и десериализации для передачи хайлатера по протоколу становятся существенными. Но несмотря на обилие различных комбинаций эффектов, которые могут быть получены изменением всех атрибутов, на деле используются их конфигурации (например красная волнистая линия, синяя закрашка для текста в ключевых словах и т.д.). Таким образом гораздо более логичным кажется передавать внутреннее знание о “типе” хайлайтера, чем все данные о его внешнем виде.

Представление элемента подсветки в протоколе

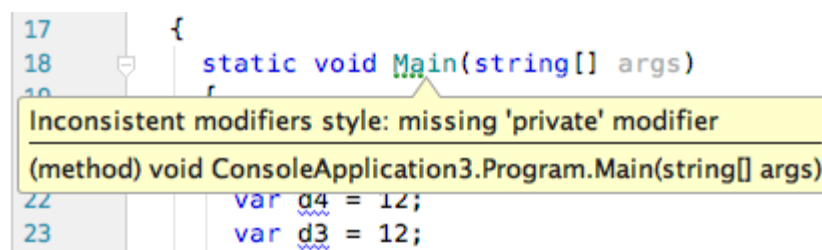
При передаче

- 1) Уникальный идентификатор
 - а) целое число, используемое для того, чтобы найти и удалить хайлайтер
- 2) Тип хайлайтера
 - а) Целое число, по которому front-end может определить его внешний вид
- 3) Абсолютное смещение начала отрезка и его конца
- 4) Версия документа
- 5) второстепенные данные для отладки и тестирования

Сообщения об ошибках и быстрая документация

Highlighter'ы, появившиеся в результате работы различных инспекций кода, сами по себе не очень информативны, их наличие в коде призвано лишь привлечь внимание программиста. Истинные причины возникновения того или иного предупреждения или ошибки становятся понятны после прочтения короткой справки (tooltip), которая появляется при наведении на мышкой на участок кода, выделенный highlighter'ом.

Кроме того почти все элементы подсветки могут показывать быструю документацию, которая помогает разработчику понять, что делает конкретный метод, для чего следует использовать данный класс или какой тип у данной переменной.



```
17 {
18     static void Main(string[] args)
19     {
22         var d4 = 12;
23         var d3 = 12;

```

Inconsistent modifiers style: missing 'private' modifier
(method) void ConsoleApplication3.Program.Main(string[] args)

(сообщения от двух highlighter'ов: совет по поводу пропущенного модификатора и полное имя класса, в котором объявлен метод)

Отображение таких сообщений – необходимая часть поддержки семантической подсветки. В то же время передача tooltip’а вместе с каждым хайлайтером вызывает значительное увеличение трафика, так как сами хайлайтеры требуют гораздо меньше данных для передачи, чем даже небольшие сообщения. Более того, если большинство сообщений об ошибках и недочетах в коде написаны достаточно ёмко и коротко, то объем документации ничем не ограничен и для некоторых методов может превышать тысячу символов.

Работа маркап-модели в условиях постоянных изменений

Структура маркап-модели

Модель семантической подсветки может быть представлена как дополнительная древовидная структура над текстом в документе. Узлами в дереве выступают отрезки, которые определяют какие эффекты должны быть применены. Большая часть таких отрезков постоянно изменяет свои границы (например, при вставки пробелов или знаков переноса строки в самом начале файла, изменяются границы абсолютно всех highlighter’ов, несмотря на то, что структура AST не изменилась). Удаление куска текста из редактора тоже приводит к изменению маркап-модели, редактор не может подсвечивать слова, если сами слова удалены, при этом все хайлайтеры, находившиеся после измененного текста опять изменяют свои границы (сдвинувшись влево на длину удаленного текста).

Таким образом, даже небольшие изменения документа могут приводить к массивным изменениям модели семантической подсветки.

В главе про выбор уровня передачи данных мы остановились на подходе, позволившем нам значительно снизить расходы на передачу одного хайлайтера, однако в условиях таких массивных изменений этого явно не

достаточно. В файле из 400 строк кода на C# содержится около 2-3 тысяч хайлайтеров, учитывая количество текстовых изменений, провоцируемых пользователем и вызовом различных опций ReSharper, работа даже в файлах такого размера была бы невозможна, при передаче всех хайлайтеров. Эвристика с передачей только тех, хайлайтеров, которые изменили свои границы тоже нисколько не поможет решить поставленную задачу (см. пример с вводом пробелом в начало документа)

Минимизация трафика

В самом деле, изменения границ хайлайтеров в соответствии с изменениями документа не требуют глубинных знаний о природе появления хайлайтеров на том или ином месте и о синтаксической структуре программы в целом. Поэтому при условии, что все хайлайтеры находятся в синхронизированном состоянии, и появлении одного изменения документа, сдвиг границ хайлайтеров может быть произведен каждой из сторон самостоятельно, без дополнительной синхронизации по протоколу.

- 1) в случае если документ изменился до начала хайлайтера, его границы должны увеличиться на разницу длин нового участка текста и того, который он собой заменил
- 2) в случае если новый текст вставили внутрь хайлайтера, то его правая граница должна увеличиться на ту же самую длину из прошлого шага
- 3) Если изменения произошли после конца хайлайтера, то его границы менять не нужно.

Поддерживая такие самокорректирующиеся структуры на каждой из сторон протокола, можно значительно сократить трафик.

При активной работе с файлом, пользователь чаще всего оставляет его в невалидном состоянии с точки зрения синтаксиса языка, поэтому почти каждое изменение сопровождается отправкой новых ошибок или их удалением в случае, если пользователь закончил синтаксически верную конструкцию или исправил ошибку. Знание о том, в какой момент нужно производить добавление или удаление хайлайтера полностью известно

только на стороне backend'a и не может быть выяснено без него. Поэтому синхронизация моделей семантической подсветки синхронизируется именно на протокольных сигналах о добавлении и удалении соответствующих хайлайтеров.

Сложности. Связь документов и модели подсветки

В “традиционных” IDE все происходит в рамках одного процесса. Здесь и далее при указании конкретных архитектурных решений будет иметься в виду `idea-community`, код которой можно найти на `github`, в прочем, общие идеи могут быть с большой вероятностью встретиться и в любых других популярных IDE. В UI-потоке осуществляется взаимодействие с пользователем (обновление содержимого документов при вводе, отрисовка элементов пользовательского интерфейса), тут же происходит работа со сложными данными, обеспечивающими различную функциональность, например, такими как деревья хайлайтеров.

При изменении документа начинает работу демон, он стартует все анализы (так называемые проходы), которые выполняются в `background`-потоках, в случае дальнейших изменения документа эти проходы инвалидируются и запускаются заново, при корректном завершении своей работы проход может вернуть хайлайтеры, которые нужно добавить в редактор, тогда поток управления передаётся в AWT, где хайлайтеры добавляются в дерево, происходит обновление UI и т.д.

Таким образом, в процессе добавления хайлайтеров в маркап модель никакие другие UI-структуры данных не могут измениться, так как поток уже занят добавлением хайлайтеров. Поэтому хайлайтеры не требуют никакой дополнительной валидации.

Синхронизация документов

В проекте Rider два процесса, у каждого из которых свой главный поток, на котором происходят обновления документов, хайлайтеров и т.д. Изменения документов в Rider'е могут быть спровоцированы каждой из сторон: front-end'ом при пользовательском вводе и back-end'ом при использовании рефакторингов или форматирования. Все операции в протоколе асинхронные, поэтому не исключена ситуация, когда пользователь вызовет форматирование, а потом успеет нажать ещё одну клавишу в тот самый момент, когда back-end уже отправил свои изменения в протокол. В этот момент содержимое документов разойдется и нужно будет разрешить конфликт в чью-то пользу. Поведение, при котором пользователь может что-то написать, а потом эти изменения исчезнут выглядит не очень интуитивным и правильным, хотя он и получит красивое форматирование, поэтому все конфликты разрешаются в пользу front-end'а, на backend отправляется обратное изменение. Так обеспечивается консистентность модели документов.

Проблемы в передаче элементов подсветки

Сложности, связанные с синхронизацией документов, и тот факт, что хайлайтеры являются результатом асинхронной работы демона, порождают ещё более сложные проблемы в процессе синхронизации highlighter'ов. Если между различными конкурентными (одновременными, двусторонними) изменениями документов проходит достаточно времени для завершения каких-нибудь анализов внутри ReSharperHost, то back-end инициирует отправку новых highlighter'ов. В случае если, пользователь успел модифицировать документ, после завершения работы анализа и до получения highlighter'а на front-end'е, то пришедший элемент расцветки может быть применён не к тому тексту. Например, если пользователь написал код, соответствующий вызову метода у представителя класса, и до того, как метод успел покраситься, добавил два пробела перед всем вызовом, полученный highlighter применится на два символа раньше (на последнюю букву названия представителя и точку), а последние две буквы навсегда останутся не покрашенными.

В отличие от приложений, в которых всё происходит в рамках одного процесса, между добавлением highlighter'а в маркап модель и отображением результата может произойти всё что угодно.

Валидация и восстановление хайлайтеров

В прошлом разделе мы описали ситуацию в которой, хайлайтеры могут быть применены не к тому тексту. Связано это с тем, что при передаче хайлайтера его знание о содержимом документа теряется, потому что документ может успеть изменить своё состояние.

Очевидным образом нужно производить валидацию и отбрасывание таких хайлайтеров, поэтому при передаче в каждый хайлайтер помещается версия документа, для которой он был создан. Это помогает определить, что версия документа помелась и теперь никто не может гарантировать, что хайлайтер получится применить к тому тексту, на котором он должен быть.

Однако ситуация становится сложнее из-за инкрементального способа синхронизации хайлайтеров. Мы не можем просто инвалидировать и выбросить хайлайтер, который не соответствует версии документа, так как backend не пришлёт нам его заново. Наглядно представим знакомый пример с вставкой пробелов и переносов строк в документ. Мы открываем файл с исходным кодом и до того, как хайлайтеры успели примениться к тексту добавляем в его начало несколько пробелов, тогда нам придется инвалидировать все не пришедшие хайлайтеры. А так как в результате ввода переводов строк и пробелов не изменилась структура файла, в маркап-моделе не появятся новых хайлайтеров. И файл остаётся полностью не раскрашенным, хотя back-end будет считать, что с ним всё в порядке.

Для того, чтобы обеспечить работу маркап модели в случае конкурентных модификаций был создан специальный механизм “проигрывания” границ хайлайтера по последним изменениям файла и специальные эвристики позволяющие отчищать историю таких изменений.

Результаты

В ходе курсовой работы были выполнены все поставленные задачи. Реализованная система используется для поддержания работы семантической подсветки, gutter-mark'ок, и областей сворачивания кода.

Система проводит корректировку и валидацию полученных highlighter'ов. Работоспособность проверена больше чем десятью тысячами пользователей.

В данный момент проект Rider находится в стадии закрытого тестирования, в которое может вступить каждый желающий и оценить качество работы и соответствие результатам, представленным в данной работе.

<https://www.jetbrains.com/rider/>