Лекция 2. Методики отладки

Баги встречаются на двух этапах жизненного цикла кода: во время разработки и в production среде. Часто ошибки, которые вылезают в течение 10-15 минут с момента написания кода, мы даже не считаем за баги – они просто часть процесса написания кода. А багами мы гораздо чаще называем проблемы, которые проявляются в production или при тестировании кода, написанного несколько дней назад; вероятно, потому что их сложнее отловить (код уже успел подзабыться). В любом случае, если код не делает того, что должен, это баг и его нужно отловить и исправить.

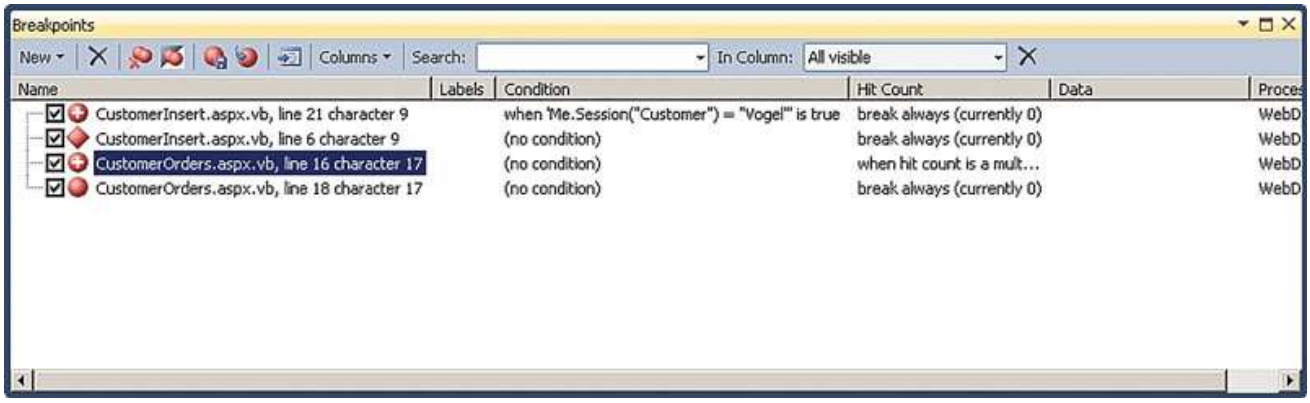
Правила эффективной отладки

- Правильные инструменты. Но инструменты бесполезны без других трех.
- Использование практик построения архитектуры и кода, отдающих предпочтение слабо связанным объектам и написанию того, что я называю Совершенно Очевидный Код – СОК (Really Obvious Code – ROC rocks!). Эти практики помогают находить баги и избегать внесения изменений, лишь добавляющих новые. И никогда не поздно перечитать *Brian W. Kernighan and P. J. Plauger's «The Elements of Programming Style» (Computing Mcgraw-Hill, 1978)*.
- Использование TDD (разработки через тестирование). Если вы разрабатываете что-то кроме сильно связанного пользовательского интерфейса и не используете TDD – вы разработчик с одной рукой, связанной за спиной. Вы менее продуктивны, чем могли бы быть, и ваш код менее надежен, чем должен быть.
- Методика отладки. Многие разработчики сразу переходят к шагу «разработка решения», что в результате редко приводит к исправлению бага, зато часто создает новые.

Методика отладки

Использование точек останова

Многие разработчики не знают всех возможностей отладки в Visual Studio, потому что отладка «и так работает». Например, хотя каждый VS-разработчик знаком с точками останова, многие не знают, что можно сделать в окне Breakpoints.



Чтобы открыть окно Breakpoins, выберите Debug | Windows | Breakpoints; в окне отобразится список всех установленных вами точек останова. Если вы не уверены, какая точка какой строке кода соответствует, просто кликните по ней двойным кликом и в редакторе откроется связанный с ней код. Определившись с нужной точкой, вы можете управлять тем, что происходит, когда она срабатывает. Я видел разработчиков, которые проверяют одни и те же переменные раз за разом, поставив точку останова в цикле. По правому клику на точке останова, выбрав When Hit (условие срабатывания), вы можете задать сообщение, которое выводится в окно Intermediate при каждом ее срабатывании. В сообщение можно включать некоторые константы: например, используя $Caller, можно вывести имя метода, вызвавшего код, содержащий точку останова. Также можно включить любые переменные, заключив их в фигурные скобки: например, {Me.NameTextBox.Text} в сообщении выведет значение поля Text. Другая возможность диалога When Hit позволяет задавать, должно ли останавливаться выполнение программы на точке останова. Если выбрать остановку, то вы увидите каждое сообщение в момент его создания; в противном случае вы сможете просмотреть все сообщения после выполнения программы. Если вы хотите остановить выполнение только при определенном условии, вы можете выбрать опции Condition или Hit Count. Опция Condition позволяет задать логическое условие, при котором произойдет остановка (например, Position > 30). Также можно выполнить остановку, если одна из переменных изменилась с момента последней остановки. Опция Hit Count прерывает выполнение только если точка останова сработала в n-й раз (или каждые n раз). Это особенно полезно, когда вам нужно остановиться где-то в конце цикла. Между прочим, мой опыт говорит, что, если в какой-то части приложения возникли проблемы, они продолжат там возникать. Если ваш опыт говорит о том же, вам понравятся дополнительные возможности Visual Studio 2010. Вы можете дать точкам останова названия, чтобы не забыть, для чего нужна каждая из них, и экспортировать их в XML-файл. В следующий раз, когда они вам понадобятся, вы можете импортировать их и начать отладку. Импорт/экспорт можно сделать с помощью тулбара вверху окна Breakpoints.

Показ и пропуск кода

Мне нравится генерирование кода (я написал книгу по тому, как это делать в .NET). Но хождение по коду, сгенерированному студией и фрэймворком, как правило не дает мне ничего полезного. А раз это не помогает найти проблему, значит делает процесс отладки менее продуктивным и лучше этого избежать.

В любой версии Visual Studio, в пункте Debug | General диалога настроек вы можете выбрать опцию Just My Code и перестать видеть код, который вы не писали. Если впоследствии вам понадобится это отключить (например, если где-то в сгенерированном коде возникает исключение), вы можете это сделать, выбрав Options and Settings в меню Debug.

Если же вы устали ходить по какой-то части вашего кода, вы можете использовать один из двух атрибутов. Поставьте на метод атрибут DebuggerHidden и вы никогда не попадете в этот метод. Если же поставить атрибут DebuggerNonUserCode, вы не будете в него попадать при включенной опции Just My Code и будете при выключенной. Я рекомендую вам использовать второй способ.

С другой стороны, если ошибка возникает где-то в коде Microsoft .NET Framework, вам может понадобится пройти не только по сгенерированному коду, но и по коду классов фрэймворка. И вы можете это сделать! Во-первых, убедитесь, что отключена опция Just My Code. Затем в диалоге Options and Settings в разделе Symbols выберите Microsoft Symbol Server (в VS 2010) или задайте путь к символам как referencesource.microsoft.com/symbols (в VS2008). Это позволит вам загрузить символы, которые поддерживают хождение по коду классов .NET. Однако вам еще нужно их загрузить. В VS2010 вы можете кликнуть кнопку Load All Symbols, но нужно набраться терпения на время скачивания. Чтобы выбрать конкретную сборку (или если вы используете VS2008), в режиме остановки процесса отладки откройте окно Modules и в списке DLL, загруженных вашим приложением, кликните правым кликом на нужной DLL и выберите Load Modules, чтобы загрузить символы для этой DLL. Конечно, подождать всё равно придется, но уже не так долго. Я один из тех, кто пишет в свойствах полезный код и хочет иметь возможность пошагово их отладить. Начиная с VS2008SP1, появилась опция (Step over properties and operators), выключающая пошаговую отладку свойств и операторов. И в VS2010 она по умолчанию включена, так что вам может понадобиться ее выключить.

Визуализация данных

Как ни странно, множество разработчиков не знакомы с визуализаторами данных в Visual Studio. Если в режиме остановки навести мышку на переменную, всплывет подсказка со значением этой переменной. Также может появиться значок с лупой – клик по нему откроет значение переменной в визуализаторе по умолчанию. Если рядом с иконкой появляется стрелка выпадающего списка, клик по стрелке покажет другие визуализаторы

для этого типа данных. Например, для строковой переменной будут показаны текстовый, XML и HTML визуализаторы. Если вы храните в строке HTML, HTML-визуализатор позволит понять, как это будет выглядеть в браузере. Визуализаторы вы можете также использовать в окнах Watch, Autos и Locals, но если вы смотрите какую-то переменную очень часто, вы можете нажать на канцелярскую кнопку в конце всплывающей подсказки, чтобы «пригвоздить» ее на этом месте. Тогда в следующий раз, когда вы будете просматривать эту часть кода, подсказка всплывет автоматически.

Кстати о подсказках – вы можете с их помощью менять значение переменной. В VS2010 даже более того: подсказку можно заставить висеть в окне, существовать всё время сеанса отладки, и даже после его окончания она будет отображать значение переменной из последнего сеанса. Однако, самые крутые инструменты (Debugger Canvas и IntelliTrace) доступны только в VS2010 Ultimate Edition.

Внешние инструменты

Visual Studio – не единственный инструмент отладки, есть сколько угодно внешних и сторонних инструментов, которые вы можете добавить в копилку. Я здесь остановлюсь только на некоторых бесплатных.

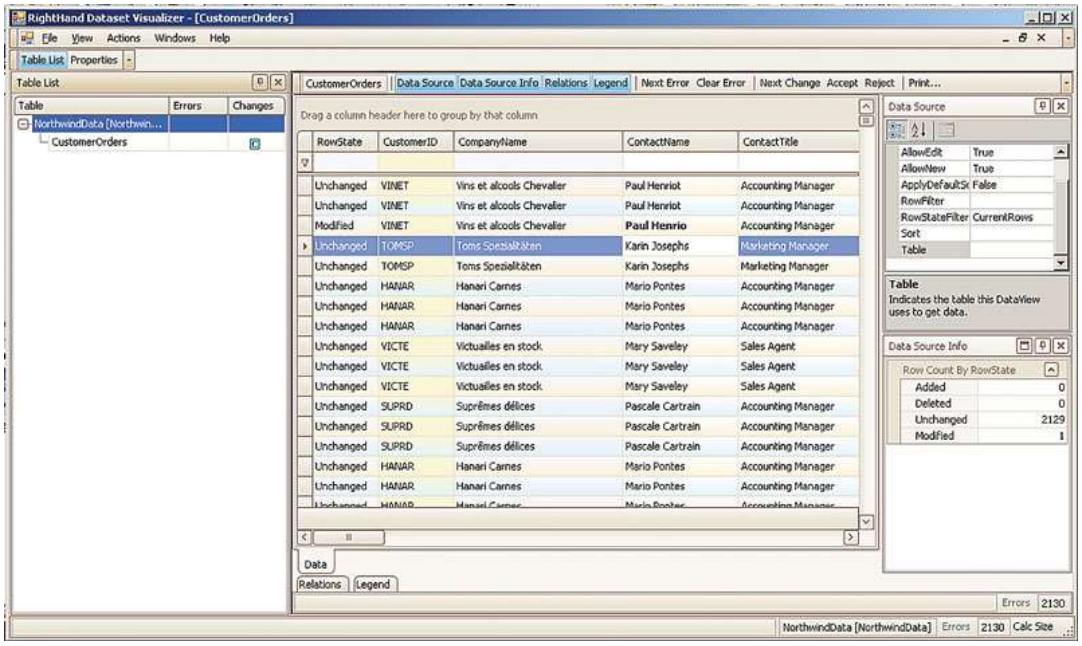
Не все внешние инструменты сделаны сторонними производителями. Если вы пишете службы Windows, то знаете, что отлаживать их непростое занятие. Хотя вы и можете подключиться к ним для отладки, к этому времени код OnStart и инициализация уже выполнится. Если баг не дает сервису запуститься, вам остается только гадать, что же пошло не так, вместо того чтобы собрать информацию для описания проблемы.

В подобной ситуации вы можете настроить Just-in-Time (JIT) отладку и автозапуск – сеанс отладки начнется, когда служба запустится или когда возникнет ошибка. Но чтобы это сделать, вам нужно воспользоваться внешними инструментами.

WinDbg: за пределами отладчика Visual Studio

Сторонние визуализаторы

Я уже говорил про визуализаторы Visual Studio, но есть еще множество сторонних. DotNetDan’s DataSet Visualizer – весьма чудесен, если вам нужно знать, что лежит в датасете (я упоминал его в блоге)



С того времени я уже обнаружил RightHand DataSet Visualizer и стал использовать его. Это MDI-приложение, которое позволяет открыть окно для каждой таблицы датасета. Кроме того, окно Relations показывает таблицы, связанные с текущей просматриваемой.

Грид, отображающий таблицу, не простой – вы можете перетащить колонку в прямоугольник вверху окна, чтобы группировать вашу таблицу по этой колонке. Также можно изменять данные в датасете и менять фильтр RowState, чтобы отображались только строки с определенным RowState (например, только удаленные). Еще можно смотреть (и менять) некоторые свойства датасета. И даже можно выгрузить датасет в XML-файл или загрузить тестовые данные из сохраненного ранее.

Следует отметить, что DotNetDan’s DataSet Visualizer быстрее загружает данные, так что я оставил его на случай, когда не нужна вся мощь RightHand.

Еще есть Web Visualizer для приложений ASP.NET. Этот визуализатор доступен на всплывающей подсказке любого объекта страницы ASP.NET (включая Me и this в коде ASP.NET)

С помощью Web визуализатора можно смотреть любые данные коллекции Server Variables объекта Server и коллекции Forms объекта Request. Также можно посмотреть строку запроса браузера и содержимое объектов Session и Application. Однако, в случае Session и Application для любых объектов, кроме скалярных данных, вы увидите только название типа объекта.

Есть и другие визуализаторы, включая те, что позволяют смотреть Cache и LINQ-запросы к Entity Framework (EF), а также позволяющие увидеть SQL на выходе LINQ-запросов к EF. Печально только то, что нет единого каталога визуализаторов. Не все, но многие можно найти через Visual Studio Extension Manager. В том числе, ASP.NET MVC Routing Visualizer. Если вы используете маршрутизацию в ASP.NET MVC или в простом ASP.NET, вам пригодится этот инструмент. Взаимодействие между правилами маршрутизации может выдавать неожиданные результаты («Почему я получаю эту страницу?»), и

отладка этих правил может быть непростой. Визуализатор позволяет вам ввести URLы и посмотреть, как маршрутизатор их декодирует, включая информацию о том, какое правило используется для каждого URL. Чтобы использовать его в режиме останова, переключитесь на ваш файл global.asax и наведите курсор на RouteTable. Когдя появится всплывающая подсказка, промотайте до коллекции Routes и кликните на значок лупы.

Трассировка

Трассировка по определению непривлекательна. Однако когда в продакшене вы нарываетесь на один из неуловимых багов, которые сложно повторить (и которые появляются и исчезают сами по себе), трассировка – единственный путь получить нужную информацию. Для этого, во-первых, вам нужно организовать запись сообщений в лог, который даст вам информацию для поимки бага. Но чтобы эта информация стала полезной, нужен инструмент для анализа содержимого лога.

Хотя в мире .NET для трассировки существует множество пакетов, я использую log4net. Среди прочих возможностей, log4net позволяет мне встроить в код отладочные сообщения и затем включать или отключать их во время работы без необходимости пересборки приложения. Одно замечание: log4net – очень гибкий инструмент и возможно больше, чем это требуется вам.

Когда дело доходит до чтения полученных логов, я использую Log Parser Lizard от Lizard Labs. В бесплатной версии некоторые возможности ограничены (цена платной – около 25$), однако мне они ни разу не понадобились. Log Parser Lizard использует SQL-подобный синтаксис для построения запросов к логам (включая файлы формата CSV и XML) и прямо из коробки понимает форматы логов IIS, событий Windows и log4net. Результаты отображаются в таблице, что делает его похожим на Server Explorer, с которым мне очень нравится работать.