**Лелеко Никита Алексеевич г.Владивосток 2011г**

11

**Leleko.CSharp.Calc**

**Библиотека для разбора математических выражений**

**http://vkontakte.ru/id5271246**

Введение

На втором курсе института у меня было задание написать приложение, выводящее введенную пользователем функцию F(x) в виде графика на языке С++. Скажу честно на тот момент я еще был слабо знаком с программированием в принципе. Наверное, это и не позволило мне найти какое-либо стороннее решение(компоненту, или использовать Java) для разбора математической функции. Пришлось изобретать свой «велосипед». В результате получился жуткий зверь о 3000 строк, который проводил разбор способом не описанным, наверное, нигде. Работал очень медленно. Но я все равно им очень гордился, тем более что никто из сокурсников даже «самокат» не смог сделать.

Уже после окончания института у меня возникла идея воплотить свой «первый» проект уже на том уровне, которого я достиг на текущий момент. До сего момента вышло уже несколько версий, к сожалению не все они сохранились для истории:

1. Август 2009г – была написана CalcScript v0.9 – умела разбирать математические выражения общего вида. Разбор проводился через два стека(Stack<T>) – операдндов и значений
2. Ноябрь 2009г – была написана уже CalcScript v2.0 – имела тот же принцип работы, но поддерживала параметры(односимвольные, латинские)
3. Июль 2010г – DynamicCalc v2.0 был совршен рывок – введена возможность кэширования для повторных вызовов
4. Январь 2011г – DynamicCalc v4.0, через Emit и DynamicMethod была добавлена возможность компиляции в IL код на лету, что еще больше ускоряло повторные вызовы
5. Январь 2012г – жажда скорости и модификации вновь заставила переписать все заного – Leleko.CSharp.Calc v2.0, собственно, про которую, и написана данная книга, имеет многочисленные улучшения и высокую производительность

Что это такое?

Что же перед вами? – Перед вами сборка(dll), в которой находятся инструменты для разбора и вычисления математических выражений времени исполнения (Runtime). Т.е. иными словами у вас есть инструмент, который может разобрать введенную функцию, скажем (2\*z\*Sin(x)/(1-y)), и вывести ее результат в зависимости от выбранных вами z,x,y, причем, для этого вам не понадобиться таскать с собой внешний компилятор, или использовать Java машину, а всего лишь добавить в свой проект небольшую (26кб) библиотеку.

На чем это работает? – Библиотека работает на NET платформе 2й и более свежих версий. Это же позволяет ей не зависеть от того установленной у Вас ОС: Windows, MacOS или даже Linux имеют рабочие Framework платформы.

Где это работает? – поскольку вся библиотека написана на NET, то с ней очень легко работать из любого поддерживающего NET языка программирования. Но(!) библиотека так же может быть зарегистрирована как COM сервер.

В чем преимущества? – некоторые люди уже написали мне, зачем ты это делаешь, есть же Maple. Мой им ответ: «найдите мне 400$ и сверните его хотя бы в 10мб». Поэтому далее я отмечу по пунктам преимущества и недостатки.

Преимущества:

1. Полная интегрируемость с NET средой безо всяких мостов типа COM или PInvoke
2. Возможность выбора трансляции или компиляции
3. Поддержка параметризации, а также параметров типа IList<double>
4. Возможность добавления пользовательских функций
5. Очень высокая скорость работы, как транслятора, так и компилятора
6. Малые размеры самой библиотеки
7. Абсолютно бесплатно

Недостатки:

1. Нет циклов
2. Нет матриц, комплексных чисел
3. Язык описания выражений

# Переменные

Поддерживаются любые латинские имена, длиной до 256 символов. Имена являются чувствительными к регистру, могут включать в себя символы верхнего/нижнего регистров, цифры от 0 до 9, однако первым в имени должен обязательно идти символ.

Маска [a-zA-Z][a-zA-z0-9]\*

# Операторы

Поддерживаемые операторы:

|  |  |
| --- | --- |
| Аддитивные | {x+y} {x-y} |
| Мультипликативные | {x\*y} {x/y} {x%y} |
| Степень | {x^y} {x\*\*y}\* |
| Унарные | {+x} {-x} |
| Логические | {x||y} {x&&y} |
| Логические унарные | {!x} |
| Операторы отношения | {x<y} {x>y} {x<=y} {x>=y} |
| Операторы равенства | {x==y} {x!=y} |
| Присваивания | {x=y} |

\* - оператор возведения в степень из наследия Fortran

(!) Операторы имеют стандартный приоритет

# Функции

Поддерживаемые функции:

|  |  |
| --- | --- |
| Модуль | Abs(x) | abs(x) |
| Арг синус | ASin(x) | Asin(x) | asin(x) |
| Арг косинус | AСos(x) | …. |
| Арг тангенс | ATan(x) | ….. |
| Синус | Sin(x) | sin(x) |
| Косинус | Cos(x) | cos(x) |
| Тангенс | Tan(x) | tan(x) |
| Гипер cинус | Sinh(x) | sinh(x) |
| Гипер косинус | Cosh(x) | cosh(x) |
| Гипер тангенс | Tanh(x) | tanh(x) |
| Степень | Pow(x,y) | pow(x) |
| Нат. логарифм | Log(x) | log(x) | Ln(x) | ln(x) |
| 10й логарифм | Log10(x) | log10(x) | Lg(x) | lg(x) |
| Максимум | Max(x,y,…) | max(x,y,…) |
| Минимум | Min(x,y,…) | min(x,y,…) |
| Умножения | Mul(x,y,…) | mul(x,y,…) |
| Сложения | Sum(x,y,…) | sum(x,y,…) |
| Корень | Sqrt(x) | sqrt(x) |
| Экспонента | Exp(x) | exp(x) |

# Константы

Поддерживаемые константы:

|  |  |
| --- | --- |
| 2.71728… | E | e |
| 3.14159… | Pi | pi |
| 0 | False | false |
| 1 | True | true |
| Бесконечнось | Inf | inf |

(!) Все константы имеют приоритет перед параметрами, т.е. можно объявить параметр с именем константы(скажем «Pi»), но вот использован он никогда не будет

# Числовые форматы

Воспринимаемые форматы чисел

|  |  |
| --- | --- |
| Целые | 1234 | -1234 |
| Вещественные\* | 1.2345| -1.2345 | 0.234 | -.234 |
| Научные | -1.234e12 | .23e+46 | 2.33e-55 |

(!) Хотя в Русской локализации NET, разделителем целой и вещественной частей по умолчанию является запятая, в данной библиотеке используется стандарт ANSI - т.е. точка (.)

1. Разбор математических выражений

(из статьи Алексея Кузнецова[1])

Те, кто занимаются различными научными расчетами или написанием научного программного обеспечения часто сталкиваются со следующей проблемой: "Каким образом добавить возможность интерактивно вводить и вычислять математические формулы в своей программе?". Традиционно существует два подхода[1]:

* "зашить" расчеты в исходный код программы
* Разрешить пользователю вводить в некотором редакторе описание задачи в виде совокупности формул, с последующей их обработкой некоторым математическим ядром

К достоинствам первого подхода можно отнести скорость выполнения и минимальные размеры исполняемого модуля (если конечно все оптимально и аккуратно запрограммировано), а также возможность реализовать сколь угодно сложные и неформализованные задачи. Но этот подход не очень гибкий, так как пользователь может настраивать только параметры задачи, а если необходимо что-либо добавить или изменить - требуется изменять исходный код программы (что чревато известными трудностями, например, любые изменения требуют тестирования и отладки программы). Второй подход можно разделить на три основных направления[1]:

* Использование спец. мат. пакетов в качестве серверов для вычисления формул
* Интерпретация
* Компиляция

Я бы, лично от себя, добавил от себя еще и 4й способ:

* Кэширование

# Трансляция - Трансляция

Интерпретация формул - достаточно распространенный подход и существует множество его реализаций. Достоинства: простота реализации, подробное диагностирование ошибок во время вычисления. Основным недостатком является крайне низкая скорость вычислений.[1]

Добавлю от себя замечание, что сама скорость вычислений отнюдь не является низкой, сравнительно низкой является скорость повторных вычислений той же функции – приходится заново разбирать всю формулу.

# Кэширование

Сама технология кэширования состоит в том, что разбираемое выражение описывается с помощью структуры, чаще всего в виде обратной польской записи(ОПЗ) или некоего дерева. И тот и другой подход имеют сравнительные преимущества и недостатки.

Вычисления выражений из ОПЗ как правило происходит значительно быстрее чем из дерева, в то же время деревья позволяют проводить гораздо более гибкую оптимизацию, например свернуть выражение ***2sin(x)cos(x)*** в простой ***sin(2x)***.

Отмечу что в моей библиотеке кэширование происходит именно в форме ОПЗ.

# Компиляция

Компиляция - анализ и трансляция формул непосредственно в машинный код или в программу на языке высокого уровня. Преобразование формул в машинный код сопряжено со значительными трудностями, так как требует от разработчика глубоких знаний в этой области и к тому же привязывает реализацию к определенной аппаратной платформе.[1]

Тут также хочу внести некоторые пояснения. С появлением NET компиляция значительно упростилась, а сам результат является кроссплатформенным. Кому будет интересна эта тема, может прочитать в msdn про System.Reflection.Emit.

1. Структура классов в сборке

## 

Литература

1. «Разбор и трансляция математических формул», Алексей Кузнецов. <http://citforum.ru/programming/delphi/delphmath/>