Gem #138: Освойте командную строку - Часть 1

Автор: Emmanuel Briot —AdaCore

Краткое содержание: Приложения можно настраивать несколькими способами. Чаще всего применяется командная строка, конфигурационные файлы и графические пользовательские интерфейсы. Технология GNAT предоставляет различные средства для сопряжения с ними: `Ada.Command_Line` и `GNAT.Command_Line`, `GNATCOLL.Config` и `GtkAda`. Последние два из них будут обсуждаться в последующих публикациях; тема данной серии - управление командной строкой.

Давайте начнём…

Приложения можно настроить несколькими способами. Среди наиболее частых-параметры командной строки, файлы конфигурации и графические пользовательские интерфейсы. Технология GNAT предоставляет различные средства для взаимодействия с ними, соответственно `Ada.Command_Line` и `GNAT.Command_Line`, `GNATCOLL.Config` и `GtkAda`.

Этот Gem связан с разбором параметров командной строки, предоставляемых вводом пользователем при запуски программы в текстовой консоли или в скрипте shell операционной системы.

Командная строка может содержать различные сведения:

- **переключатели**, которые могут быть короткими переключателями (например, «`-a`») или длинными переключателями (например, «`--long`»). Эти переключатели могут иметь дополнительные параметры, которые идут сразу после самого переключателя и могут быть разделены различными символами, например, «`-a значение`» или «`--long = значение`». Приложения часто позволяют сочетать короткие переключатели (например, «`-ab`» совпадает с «`-a -b`»). Эта комбинация может включать поиск общего префикса для ключей (в gnatmake «`-gnatpQ`» эквивалентен «`-gnatp -gnatQ`»). Также возможно иметь псевдонимы. Например, в gnatmake пользователи могут использовать «`-gnaty`» вместо гораздо более длинного «`-gnaty3abcefhiklmnprst`»).

- **аргументы**, такие как имена файлов для манипуляции, которые не связаны с переключателями. Одна из трудностей при разборе командной строки заключается в различении параметров переключателя и других аргументов.

- **секции**, которые обычно используются для предоставления переключателей для других приложений, порожденных основным. Хорошим примером этого является командная строка `gnatmake`, где можно указать переключатели для компилятора (после «`--cargs`»), для компоновщика (после «`--largs`») и так далее. В таком разделе переключателей первое приложение, как правило, не будет знать, какие переключатели являются допустимыми, поэтому оно должно разрешать любые переключатели или аргументы, чтобы затем передавать их другому приложению.

Это богатый набор возможностей, и синтаксический анализ командной строки может привести к тому, что код будет трудно поддерживать и может затруднить добавление новых ключей.

Пользователи также ожидают наличия нескольких стандартных ключей, среди которых «`-h`» и «`--help`», которые должны предоставить краткое описание всех возможных ключей. Это не заменяет надлежащую документацию, но действует как краткое резюме и напоминание.

Стандарт Ada предоставляет пакет `Ada.Command_Line`. Этот пакет является удобным способом доступа к каждому из элементов в командной строке, но он не предоставляет никакой

помощи в описании их значения. Приложение отвечает за определение того, есть ли у него переключатель, аргумент для переключателя, который вы видели ранее, или аргумент.

Пакет `GNAT.Command_Line` построен поверх этого и предоставляет гораздо более богатый API. Этот API можно использовать на нескольких уровнях абстракции, которые мы сейчас опишем, начиная с самого низкого уровня до более высоких уровней.

Простой синтаксический анализ командной строки

Начнем с примера кода, который анализирует командную строку.

```
with GNAT.Command_Line;   use GNAT.Command_Line;
with Ada.Text_IO;         use Ada.Text_IO;

procedure Main is
begin
   loop   -- 1
      case Getopt ("a b: -long= -help") is    -- 2
         when 'a' =>
            Put_Line ("Seen -a");   -- 3
         when 'b' =>
            Put_Line ("Seen -b with arg=" & Parameter);  -- 4
         when '-' =>
            if Full_Switch = "--long" then   -- 5
               Put_Line ("Seen --long with arg=" & Parameter);
            elsif Full_Switch = "--help" then
               Put_Line ("Seen --help");
            end if;
         when others =>  -- 6
            exit;
      end case;
   end loop;

   Put_Line ("File argument was " & Get_Argument);  -- 7
end Main;
```

Разбор командной строки выполняется в цикле (шаг 1), который просматривает все элементы командной строки, чтобы определить, является ли он переключателем, параметром или аргументом.

Вызов `Getopt` на шаге 2 - то, где большая часть работы сделана. Он будет смотреть на следующий элемент в командной строке и извлекать из него информацию. Обязательный аргумент - это описание списка допустимых ключей. Синтаксис полностью описан в `g-comlin.ads` (или доступен через меню / Help / GNAT Runtime в GPS), но в основном сводится к следующему: предполагается, что все переключатели начинаются с одного и того же символа (по умолчанию '-')), который не нужно повторять здесь. В этом примере приложение принимает четыре переключателя. Первый - «`-a`», который не принимает параметров; вторая - «`-b`», для которой требуется параметр (между переключателем и его параметром может быть пробел, поэтому допустимы «`-bparam`» и «`-b param`»); третий ключ - «`--long`», для которого также требуется параметр, и он должен быть отделен от ключа либо пробелом, либо знаком равенства. Наконец, последний параметр - «`--help`», который не принимает параметров.

Для каждого параметра, который он находит в командной строке, `Getopt` возвращает свой первый символ, чтобы обеспечить быстрый оператор `case`, а не последовательность операторов `if`. К сожалению, этот дизайн не совсем подходит для длинных переключателей, которые начинаются с «`-`» и, следовательно, требуют операторов `if`, как в шаге 5.

Шаг 4 печатает параметр, который был передан для "-b". Параметр - это функция, которая возвращает параметр текущего переключателя.

Когда в командной строке больше нет переключателей, Getopt возвращает ASCII.NUL, и шаг 6 выходит из цикла.

Наконец, мы предполагаем, что приложению требуется одно имя файла в командной строке. Это имя файла может быть получено путем вызова Get_Argument. Эта функция не может быть вызвана до тех пор, пока мы не найдем все переключатели, поскольку в противном случае между параметрами и аргументами переключателя могут быть неоднозначности. Get_Argument предоставляет еще одну дополнительную услугу: он может расширять аргумент, содержащий метасимвол, например «*», например «*.adb». В системах Unix это уже было расширено оболочкой, что привело к множеству аргументов. Но в Windows само приложение отвечает за это расширение, и Get_Argument может сделать это за вас.

Как таковой, код будет автоматически корректно работать со следующей командной строкой: "-a -bvalue --long = value" или даже "-abvalue". Однако командная строка, такая как «-c» или «-avalue», будет отклонена, и возникнет исключение.

Разделы

Как мы уже упоминали, такие приложения, как gnatmake, принимают разные наборы переключателей для различных приложений, которые они порождают. Такие разделы должны быть объявлены как таковые:

```
Initialize_Option_Scan (Section_Delimiters => "cargs bargs largs");
```

На основе Getopt необходимо несколько циклов, по одному для каждого раздела. Таким образом, код будет похож на:

```
Goto_Section ("cargs");
--  a loop around Getopt as described above

Goto_Section ("bargs");
--  a loop around Getopt as described above
```

Вполне вероятно, что параметр Getopt в одном или нескольких разделах будет начинаться с "*", чтобы указать, что все элементы, найденные в командной строке в этом разделе, должны быть возвращены как есть, не пытаясь угадать, являются ли они переключателями или аргументами, потому что gnatmake просто хочет передать их как есть приложению, которое оно порождает.

Имитация командной строки

В некоторых системах, особенно встраиваемых, отсутствует понятие командной строки. Поэтому GNAT.Command_Line будет казаться бесполезным в этих системах.

Но на самом деле, он не должен читать элементы из реальной командной строки. Он также может читать их из массива строк. Таким образом, используя тот же пример, что и выше, мы можем сохранить большую часть кода и просто смоделировать командную строку. Например:

```
with GNAT.Command_Line;   use GNAT.Command_Line;
with GNAT.OS_Lib;   use GNAT.OS_Lib;

...

declare
   Parser : Opt_Parser;
```

```
      Command_Line : constant Argument_List_Access :=
         Argument_String_To_List ("-a -bvalue --long=value");
   begin
      Initialize_Option_Scan (Parser, Command_Line);

      loop
         case Getopt ("-a -b: -long= -help", Parser => Parser) is
           ... as before ...
         end case;
      end loop;

      Free (Parser);
   end;
```

`Command_Line` может быть пустым списком в случае, когда мы хотим проанализировать фактические элементы командной строки (при условии, что система поддерживает командные строки).

В следующем Gem этой серии будут обсуждаться аспекты высокоуровневого API `GNAT.Command_Line`, которые значительно упрощают обработку в командной строке.

Связанный со статьёй текст программы

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Автор: Emmanuel Briot - AdaCore

Эммануэль Брио работает в AdaCore с 1998 года. Он принимал участие в различных проектах, в частности, ориентированных на графические пользовательские интерфейсы, включая GtkAda, GPS, XML / Ada, GnatTracker и нашу внутреннею CRM. Он получил степень инженера в Национальной школе телекоммуникаций - Брест, Франция (Ecole Nationale des Telecommunications - Brest, France).

Last Updated: 10/13/2017
Posted on: 12/10/2012

Обсуждение…