Gem #52: Возможности создания скриптов в GNAT (Часть 1)

Автор: Emmanuel Briot, AdaCore

Краткое содержание: Gem Ada #52 - По мнению большинства программистов, все языки программирования можно разделить на две категории: языки описания сценариев (скриптов), и все остальные. Трудно назвать одно определяющее различие между этими двумя категориями. Язык относят к той или иной категории чаще всего на основе того, каких масштабов приложения на нем пишутся, является ли он интерпретируемым или компилируемым, и насколько легко производить такие операции как манипуляция внешними процессами.

Однако, с уверенностью можно заявлять, что Ada не относится к скриптовым языкам. Несмотря на это, в данной статье Gem описываются несколько пакетов сборки GNAT, с помощью которых можно выполнять задачи, связанные с применением скриптов, которые считаются возможными только в языках, специализирующихся на скриптах. Конечно же, переносимость программ - одно из важнейших преимуществ языка Ada!

Давайте начнем…

Первое, что обычно требуется программе, - разобрать свою командную строку, чтобы узнать, какие функции пользователь хочет активировать. Стандартная Ada (имеется в виду пакеты, включённые в стандарт Ada 2005) предоставляет пакет `Ada.Command_Line`, который в основном дает вам доступ к каждому аргументу в командной строке. Но GNAT предоставляет гораздо более продвинутый пакет `GNAT.Command_Line`, который помогает манипулировать этими аргументами, чтобы вы могли легко извлекать ключи, их (возможно, необязательные) аргументы и любые оставшиеся аргументы. Вот краткий пример использования этого пакета:

```
with GNAT.Command_Line;  use GNAT.Command_Line;
procedure Main is
begin
  loop
     case Getopt ("h f: d? e g") is
       when 'h' => Display_Help;
       when 'f' => Set_File (Parameter);
       when 'd' => Set_Directory (Parameter);
       when 'e' | 'g' => Do_Something (Full_Switch);
       when others => exit;
     end case;
  end loop;
exception
  when Invalid_Switch | Invalid_Parameter =>
     Display_Help;
end Main;
```

Это приложение принимает несколько переключателей, так называемых «ключей». Ключ `-f` требует дополнительного аргумента, тогда как ключ `-d` имеет необязательный аргумент. Приложение можно вызвать несколькими способами: например, «`-ffile -e -g`», «`-f file -e -g`» или «`-f file -eg`». `GNAT.Command_Line` является чрезвычайно гибким в том, что он принимает от пользователя, что может помочь сделать приложение более простым в использовании.

Другим удобным пакетом является `GNAT.Regpat`, который обеспечивает возможность обработки регулярных выражений в программах Ada. Этот пакет уже обсуждался в предыдущем Gem, связанном с поиском, поэтому мы не будем здесь подробно останавливаться (см. Gem # 25).

Часто бывает, что скрипты должны разбирать текстовые файлы. Хотя стандартный пакет `Ada.Text_IO` предоставляет доступ к таким файлам, он имеет несколько недостатков. Во-первых, он довольно медленный. Фактически, стандарт Ada заставляет этот пакет управлять несколькими дополнительными элементами информации внутри пакета, что может привести к значительным

накладным расходам. Кроме того, файл считывается кусками из цепочки из блоков, такой подход на большинстве систем медленный (и требует много блокирующих системных вызовов). Там, где это возможно, лучше использовать более эффективные стандартные пакеты, такие как `Stream_IO`. Кроме того, `Ada.Text_IO` не предоставляет возможности синтаксического анализа.

Компоненты `GNAT Reusable` (первоначально выпущенные клиентам в июле 2008 года) предоставляют пакет `GNATCOLL.Mmap`. Этот пакет обычно использует более эффективные системные вызовы для чтения файла и приводит к гораздо большей скорости, если ваше приложение читает множество текстовых файлов.

```
with GNATCOLL.Mmap;  use GNATCOLL.Mmap;
procedure Main is
   File : Mapped_File := Open_Read ("filename");
   Str  : Str_Access;
begin
   Read (File);  --  read whole file at once
   Str := Data (File);
   --  you are now manipulating an in-memory version of the file
   Close (File);
end Main;
```

Этот пакет также предоставляет поддержку для того, чтобы только считать сразу часть файла в память. Такая возможность важна, если Вы управляете огромными по размерам файлами, которые не могут целиком отображаться в памяти. Но чаще всего более эффективно просто считать целый файл сразу.

Относительно средств обработки-разбора, так называемого «парсинга», другой пакет доступен программистам: `GNAT.AWK`. Этот пакет обеспечивает интерфейс, подобный стандарту awk утилите в системах Unix. В частности один из его режимов - неявный цикл по целому файлу. Сопоставление с образцом применено к каждой строке, по очередно, и когда образец соответствует, производится вызов так называемой процедуры обратного вызова (которая должна быть за ранее зарегистрирована). Это означает, что Вы больше не должны делать весь парсинг-разбор самостоятельно. Обеспечены несколько типов так называемых итераторов (iterators), но этот Gem иллюстрирует только один. Посмотрите интерфейс `g-awk.ads` для большего количества примеров.

```
with GNAT.AWK;  use GNAT.AWK;
procedure Main is
   procedure Action1 is
   begin
      Put_Line (Field (2));
   end Action2;

   procedure Default_Action is
   begin
      null;
   end Default_Action;
begin
   Register (1, "saturn|earth", Action1'Access);
   Register (1, ".*", Default_Action'Access);
   Parse (";", "filename");
end Main;
```

Каждая строка в файле, обозначенная «`filename`», содержит группы полей с точкой с запятой, используемой для разделения групп. Действия регистрируются для разных значений первого поля в группе. Когда первое поле соответствует «`saturn`» или «`earth`», вызывается соответствующее действие, а второе поле печатается. Для всех других планет ничего не делается. Этот код, безусловно, больше, чем эквивалентная команда оболочки Unix, использующая `awk`, но по-прежнему имеет очень приемлемый размер. Конечно, если вы ошибаетесь в своей программе, очень вероятно, что компилятор поможет вам найти его (Вопрос: Вы когда-либо пытались отладить 20-

строчную `awk`-программу, не говоря уже о более длинных программах, где ограничители быстро стали проблемой?)

Предстоящий Gem обсудит дополнительные возможности сценариев, сосредоточившись на манипулировании внешними процессами.

Связанный со статьёй текст программы

Attached Files отсутствуют

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

Об авторе

Сведения отсутствуют.

Last Updated: 10/13/2017
Posted on: 11/10/2008

Обсуждение…