Gem #20: Использование прагмы Shared_Passive для обеспечения сохранности данных

Автор: Pascal Obry, EDF R&D

Краткое содержание: Gem Ада #20 - Использование прагмы Shared_Passive для обеспечения сохранности данных.

Давайте начнем…

Сохраняемость данных может быть достигнута многими способами, например, начиная с простого, как использование рукописного кода для хранения и загрузки данных в некоторые текстовые файлы, до чего-то столь сложного, как сопоставление данных в реляционной или объектной базе данных.

В некоторых случаях мы просто хотим сохранить содержимое набора переменных. Ada оказывает такую поддержку с использованием прагмы "Shared_Passive" приложения-E (Annex-E). Давайте напишем простой счетчик, который будет увеличиваться при каждом запуске приложения:

```
package Store is
   pragma Shared_Passive;
   Counter : Natural := 0;
end Store;
```

И да, это все! Текущее значение переменной считывается во время элаборации (elaboration) и записывается во время завершения программы. Общее состояние пассивного блока сохраняется на диске с использованием индивидуального файла для каждого объявления верхнего уровня (variables, protected objects). Имя файла состоит из имени элемента и имени декларации, разделенного точкой. Например, вышеуказанное состояние переменной счетчика будет сохранено в файле с именем «store.counter».

Поэтому это так просто использовать в процедуре `Main`, как например:

```
with Ada.Text_IO;
with Store;

procedure Main is
   use Ada.Text_IO;
begin
   Put_Line ("Counter : " & Natural'Image (Store.Counter));
   Store.Counter := Store.Counter + 1;
end Main;
```

Каждый раз при запуске программы счетчик увеличивается на единицу. Нет ничего в программе, чтобы явно сохраняло или загружало Counter.

В контексте параллельного программирования может потребоваться добавить правильную синхронизацию. Это можно легко сделать с помощью защищенного объекта в общем пассивном пакете.

```
package Store is
   pragma Shared_Passive;

   protected Shared is
      function Counter return Natural;
```

```
      procedure Increment;
   private
      C : Natural := 0;
   end Shared;
end Store;
```

Обратите внимание, что набор объектов, которые могут быть объявлены, ограничен, поскольку общий пассивный блок может зависеть только от чисто пассивной единицы или других общих пассивных единиц. Так, например, невозможно объявить Unbounded_String или любые Ada.Containers в совместно используемом пассивном блоке.

Однако в общей пассивной секции можно объявить сложные объекты, такие как записи или массивы, и сохранить их автоматически. Давайте возьмем для примера следующую комплексную матрицу:

```
package Store is
   pragma Shared_Passive;

   type Complex is record
      X, Y : Float;
   end record;

   type Matrix is array
      (Positive range <>, Positive range <>) of Complex;

   M : Matrix (1 .. 3, 1 .. 3);

end Store;
```

Несмотря на ограничения, эта недорогая, в части написания программистом кода программы, поддержка сохранения состояния может быть довольно удобной при некоторых обстоятельствах.

Связанный исходный код

```
package Store is
   pragma Shared_Passive;

   protected Shared is
      function Counter return Natural;
      --  Returns current counter value

      procedure Increment;
      --  Increment counter by one

      procedure Set (Name : in String);
      --  Store Name into an internal buffer

      function Get return String;
      --  Returns last stored name

   private
      C      : Natural := 0;
      Buffer : String (1 .. 100) := "undefined" & (1 .. 91 => ' ');
      Last   : Natural := 9;
   end Shared;

end Store;

package body Store is
```

```
   ------------
   -- Shared --
   ------------

   protected body Shared is

      -------------
      -- Counter --
      -------------

      function Counter return Natural is
      begin
         return C;
      end Counter;

      ---------------
      -- Increment --
      ---------------

      procedure Increment is
      begin
         C := C + 1;
      end Increment;

      ---------
      -- Set --
      ---------

      procedure Set (Name : in String) is
      begin
         if Name'Length > Buffer'Length then
            Buffer := Name (Name'First .. Name'First + Buffer'Length - 1);
            Last := Buffer'Last;
         else
            Buffer (1 .. Name'Length) := Name;
            Last := Name'Length;
         end if;
      end Set;

      ---------
      -- Get --
      ---------

      function Get return String is
      begin
         return Buffer (1 .. Last);
      end Get;

   end Shared;

end Store;

with Ada.Calendar.Formatting;
with Ada.Text_IO;
with Store;

procedure Main is
   use Ada;
   use Ada.Text_IO;
begin
   Put_Line
     ("Counter : " & Natural'Image (Store.Shared.Counter)
        & " last run " & Store.Shared.Get);
   Store.Shared.Increment;
```

```
   Store.Shared.Set (Calendar.Formatting.Image (Calendar.Clock));
end Main;
```

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

Обсуждение…

1. 4-го декабря 2007 в 9:33

Christoph Grein сказал:

Эй, это - действительно хороший GEM, на самом деле. Два комментария, все же.

Хранение в файле, я считаю, деталь реализации GNAT (по крайней мере, имя файла). На это нужно указать. AARM E.1 (11.a)

Где в RM это определено, что инициализация совместно используемой пассивной переменной только выполнена в самый первый раз? Я искал в RM, но без пользы, не нашел..

2. 4-го декабря 2007 в 18:57

Paul F. Pearson сказал:

В каком формате хранятся данные? Я предполагаю, что это - по существу потоковый вывод (например, `'Output`)? Если так, могло бы быть интересно, если `'Output` мог бы быть переопределен (предполагая, что Ada.Streams – Pure, который я бы не рассматривал).

Спасибо за GEM-20!

3. 6-го декабря 2007 в 19:36

Pascal Obry сказал:

Christoph,

Спасибо за добрые слова. Реализация GNAT должна действительно быть определена четко.

Паскаль.

4. 6-го декабря 2007 в 19:40

Pascal Obry сказал:

Paul, Да Вы правы, что вывод - просто поток. И это действительно возможный определить переданный потоком формат. Мы могли соедините этот GEM-020 с предыдущим Gem-19 в использование формата XML.

Например, следующая реализация хранилища будет:

```
with Ada.Streams;
package Store is
  pragma Shared_Passive;

  use Ada.Streams;
```

```
  type My_Int is new Integer;

  procedure Write_Int
    (S : access Root_Stream_Type'Class; V : in My_Int);
  for My_Int'Write use Write_Int;

  Counter : My_Int := 0;
end Store;

package body Store is
  procedure Write_Int
    (S : access Root_Stream_Type'Class; V : in My_Int) is
  begin
    String'Write
      (S, "" & My_Int'Image (V) & "");
  end Write_Int;
end Store;
```

Реализацию Read оставляю читателю как упражнение.

Pascal.

5. 6-го декабря 2007 в 21:48

Paul F. Pearson сказал:

Pascal, Спасибо. Я не читал о Приложении E прежде, чем прочитать GEM-20. Теперь моя голова болит.:-)