Gem #37: Пакет кольцевых буферов в иерархии GNAT (Часть 2)

Автор: Pat Rogers, AdaCore

Краткое содержание: Gem Ada #37 - В части 1 настоящего Gem (Gem Ada #35) были кратко представлены кольцевые буферы, защищённые типы, и объявление обобщённого пакета GNAT. `Bounded_Buffers`, экспортирующего защищённый тип `Bounded_Buffer`. В части 2 исследуется скрытая часть `Bounded_Buffer` и реализация защищённых записей и функций.

Давайте начнем…

Чтобы резюмировать Часть 1, напомним, защищённые типы добавляют эффективный стандартный блок к средствам управления задачами Ada. Взаимное исключение автоматически обеспечено, и синхронизация условия сразу выражена через барьеры входа, все без расхода на переключения контекста задачи. Таким образом, защищённые типы являются естественным подходом реализации для кольцевого ограниченного буфера, используемого в контексте параллельного программирования. Полное объявление универсального пакета и инкапсулированный защищённый Тип, приводятся ниже по тексту, включая теперь приватную часть. (Мы пропустили комментарии в коде, так как они присутствуют в описании.)

```
with System;

generic
   type Element is private;
package GNAT.Bounded_Buffers is
   pragma Pure;

   type Content is array (Positive range <>) of Element;

   Default_Ceiling : constant System.Priority := System.Default_Priority;

   protected type Bounded_Buffer
      (Capacity : Positive;
       Ceiling : System.Priority)
   is
      pragma Priority (Ceiling);

      entry Insert (Item : Element);
      entry Remove (Item : out Element);

      function Empty  return Boolean;
      function Full   return Boolean;
      function Extent return Natural;

   private
      Values   : Content (1 .. Capacity);
      Next_In  : Positive := 1;
      Next_Out : Positive := 1;
      Count    : Natural  := 0;
   end Bounded_Buffer;

end GNAT.Bounded_Buffers;
```

Приватная часть типа, `Bounded_Buffer` содержит инкапсулированные данные, к которым можно только получить доступ через экспортируемые подпрограммы Абстракция - это ограниченный буфер, и для хранения значений буфера он использует значения компонентов типа содержимого контента. Дискриминантная способность определяет фактическая верхняя граница элемента массива, который, как различные объекты `Bounded_Buffer` могут иметь разную мощность. Для кольцевого буфера требуются два индекса массива, один из которых указывает, куда будет помещено следующее вставленное значение, а другой-указывает, откуда придет следующее

изымаемое из буфера значение. Эти индексы - компоненты Next_In и Next_Out, соответственно. Наконец, каждый объект Bounded_Buffer содержит количество числа элементов, в настоящее время сохранённого буфером. Этот счётчик используется для обеспечения синхронизации условий, необходимой для предотвращения вставки элементов в полный буфер или удаления элементов из пустого буфера. Существуют и другие способы определить, является ли буфер полным или пустым, но этот подход, основанный на методе подсчёта, ясен, а также полезен для функции Extent.

Тело универсального (шаблона - generic package body) пакета содержит только тело защищённого типа, подобно тому, как защищённое тело содержит только тела экспортируемых записей и функций, поэтому мы опускаем строки для деклараций пакета и тела защищённого типа и вместо этого фокусируем внимание на сами подпрограммы.

Первая процедура в защищённом теле - это первая запись в Insert. Обратите внимание на барьер «Count /= Capacity», обеспечивающий синхронизацию состояния для состояния «не заполнено» (“not full”). Абоненты для вставки сохраняются до тех пор, пока барьер не станет True. Как и в случае любой защищённой записи или защищённой процедуры, вызывающие абоненты автоматически выполняют взаимоисключающий доступ к инкапсулированным данным.

```
entry Insert (Item : Element) when Count /= Capacity is
begin
   Values (Next_In) := Item;
   Next_In := (Next_In mod Capacity) + 1;
   Count := Count + 1;
end Insert;
```

Тело элемента Insert простое. Обратите внимание, что у нас есть индекс Next_In, который охватывает допустимые значения индекса на основе ёмкости. Мы не могли использовать модульный тип для любого индекса, поскольку для этого требовалось бы статическое значение для модуля, но у нас есть только дискриминант Capacity.

Тело для ввода Remove аналогично элементу Insert, с ожидаемыми различиями.

```
entry Remove (Item : out Element) when Count > 0 is
begin
   Item := Values (Next_Out);
   Next_Out := (Next_Out mod Capacity) + 1;
   Count := Count - 1;
end Remove;
```

Важный момент для каждой Entry (точки входа и синхронизации) - обработка компонента Count. Когда любая обработка завершается, другая потенциальная операция с буфером ожидает проверяя стал ли барьер истиной. Если барьер принимает значение истина, то вызывающей стороне ожидания (если таковые имеются) позволяют выполниться её обработку, и процесс повторяется. Таким образом очень простое выражение условия обеспечено барьерами, и они автоматически переоценены каждый раз, когда они могли возможно измениться.

Тела функции ещё более просты, чем логика работы точки входа и синхронизации (Entry). Стоит повторить мнение, высказанное в части 1 этого Gem (Gem Ada #35), а именно, что состояние защищённого объекта может сразу измениться после вызова к таким функциям.

```
function Empty return Boolean is
begin
   return Count = 0;
end Empty;

function Full return Boolean is
begin
   return Count = Capacity;
end Full;
```

```
   function Extent return Natural is
   begin
      return Count;
   end Extent;
```

Клиенты могут запрашивать ёмкость любого объекта `Bounded_Buffer`, просто читая дискриминант `Capacity`. Дискриминант не может быть изменён, поэтому эффект почти такой же, как результат реализации запроса через функцию.

Как писал Никлаус Вирт (цитируя Hoare), как только вы правильно структурируете данные, код просто пишет сам. По общему признанию, он не использовал эти точные слова, но я думаю, вы согласитесь, что структуры данных, используемые в типе `Bounded_Buffer`, в сочетании с семантикой защищенных типов, делают тела подпрограмм тривиальными для записи.

Связанный исходный код из части 1

Attached Files из Gem Ada #35

```
with GNAT.Bounded_Buffers;

with Ada.Text_IO;    use Ada.Text_IO;

procedure Demo_Buffers is

   type Job is new Integer;  --  for example

   Stop : constant Job := -1;  --  arbitrary

   package Jobs is new GNAT.Bounded_Buffers (Job);

   Buffer_Size : constant := 20;  --  arbitrary
   Iterations  : constant := Buffer_Size * 3;  --  arbitrary
      --  we want to insert more items than the buffer can
      --  hold at one time to ensure that the condition
      --  synchonization is exercised

   Buffer : Jobs.Bounded_Buffer (Capacity => Buffer_Size,
                                 Ceiling  => Jobs.Default_Ceiling);

   task Producer;

   task Consumer;


   task body Producer is
   begin
      for K in Job range 1 .. Iterations loop
         --  do some work to produce the next value;
         --  in this case we just insert the loop parameter
         --  ...
         Buffer.Insert (K);
         --  ...
         --  do some more work
         --  ...
      end loop;

      Buffer.Insert (Stop);
   end Producer;
```

```
   task body Consumer is
      Next : Job;
   begin
      loop
         Buffer.Remove (Next);
         exit when Next = Stop;
         --  process the next job ...
         Put_Line (Next'Img);  --  for example
         Flush;
      end loop;
   end Consumer;

begin
   null;
end Demo_Buffers;
```

Файлы из GNAT LIBRARY COMPONENTS

```
 1. ------------------------------------------------------------------------------
 2. --                                                                          --
 3. --                         GNAT LIBRARY COMPONENTS                          --
 4. --                                                                          --
 5. --                 G N A T . B O U N D E D _ B U F F E R S                  --
 6. --                                                                          --
 7. --                                 S p e c                                  --
 8. --                                                                          --
 9. --                    Copyright (C) 2003-2010, AdaCore                      --
10. --                                                                          --
11. -- GNAT is free software;  you can  redistribute it  and/or modify it under --
12. -- terms of the  GNU General Public License as published  by the Free Soft- --
13. -- ware  Foundation;  either version 3,  or (at your option) any later ver- --
14. -- sion.  GNAT is distributed in the hope that it will be useful, but WITH- --
15. -- OUT ANY WARRANTY;  without even the  implied warranty of MERCHANTABILITY --
16. -- or FITNESS FOR A PARTICULAR PURPOSE.                                     --
17. --                                                                          --
18. --                                                                          --
19. --                                                                          --
20. --                                                                          --
21. --                                                                          --
22. -- You should have received a copy of the GNU General Public License and    --
23. -- a copy of the GCC Runtime Library Exception along with this program;     --
24. -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
25. -- <http://www.gnu.org/licenses/>.                                          --
26. --                                                                          --
27. -- GNAT was originally developed  by the GNAT team at  New York University. --
28. -- It is now maintained by Ada Core Technologies Inc (http://www.gnat.com). --
29. --                                                                          --
30. ------------------------------------------------------------------------------
31.
32. --  This package provides a thread-safe generic bounded buffer abstraction.
33. --  Instances are useful directly or as parts of the implementations of other
34. --  abstractions, such as mailboxes.
35.
36. --  Bounded_Buffer is declared explicitly as a protected type, rather than as
37. --  a simple limited private type completed as a protected type, so that
38. --  clients may make calls accordingly (i.e., conditional/timed entry calls).
39.
40. with System;
41.
42. generic
43.    type Element is private;
44.    --  The type of the values contained within buffer objects
45.
46. package GNAT.Bounded_Buffers is
47.    pragma Pure;
```

```
48.
49.    type Content is array (Positive range <>) of Element;
50.    --  Content is an internal artefact that cannot be hidden because protected
51.    --  types cannot contain type declarations.
52.
53.    Default_Ceiling : constant System.Priority := System.Default_Priority;
54.    --  A convenience value for the Ceiling discriminant
55.
56.    protected type Bounded_Buffer
57.       (Capacity : Positive;
58.       --  Objects of type Bounded_Buffer specify the maximum number of Element
59.       --  values they can hold via the discriminant Capacity.
60.
61.       Ceiling : System.Priority)
62.       --  Users must specify the ceiling priority for the object. If the
63.       --  Real-Time Systems Annex is not in use this value is not important.
64.    is
65.       pragma Priority (Ceiling);
66.
67.       entry Insert (Item : Element);
68.       --  Insert Item into the buffer, blocks caller until space is available
69.
70.       entry Remove (Item : out Element);
71.       --  Remove next available Element from buffer. Blocks caller until an
72.       --  Element is available.
73.
74.       function Empty return Boolean;
75.       --  Returns whether the instance contains any Elements.
76.       --  Note: State may change immediately after call returns.
77.
78.       function Full return Boolean;
79.       --  Returns whether any space remains within the instance.
80.       --  Note: State may change immediately after call returns.
81.
82.       function Extent return Natural;
83.       --  Returns the number of Element values currently held
84.       --  within the instance.
85.       --  Note: State may change immediately after call returns.
86.
87.    private
88.       Values   : Content (1 .. Capacity);
89.       --  The container for the values held by the buffer instance
90.
91.       Next_In  : Positive := 1;
92.       --  The index of the next Element inserted. Wraps around
93.
94.       Next_Out : Positive := 1;
95.       --  The index of the next Element removed. Wraps around
96.
97.       Count    : Natural  := 0;
98.       --  The number of Elements currently held
99.    end Bounded_Buffer;
100.
101. end GNAT.Bounded_Buffers;
```

```
1. ------------------------------------------------------------------------------
2. --                                                                          --
3. --                         GNAT COMPILER COMPONENTS                         --
4. --                                                                          --
5. --                 G N A T . B O U N D E D _ B U F F E R S                  --
6. --                                                                          --
7. --                                 B o d y                                  --
8. --                                                                          --
9. --                    Copyright (C) 2003-2010, AdaCore                      --
10. --                                                                         --
```

```
11. -- GNAT is free software;  you can  redistribute it  and/or modify it under --
12. -- terms of the  GNU General Public License as published  by the Free Soft- --
13. -- ware  Foundation;  either version 3,  or (at your option) any later ver- --
14. -- sion.  GNAT is distributed in the hope that it will be useful, but WITH- --
15. -- OUT ANY WARRANTY;  without even the  implied warranty of MERCHANTABILITY --
16. -- or FITNESS FOR A PARTICULAR PURPOSE.                                     --
17. --                                                                         --
18. --                                                                         --
19. --                                                                         --
20. --                                                                         --
21. --                                                                         --
22. -- You should have received a copy of the GNU General Public License and    --
23. -- a copy of the GCC Runtime Library Exception along with this program;     --
24. -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
25. -- <http://www.gnu.org/licenses/>.                                          --
26. --                                                                         --
27. -- GNAT is maintained by Ada Core Technologies Inc (http://www.gnat.com).   --
28. --                                                                         --
29. -------------------------------------------------------------------------------
30.
31. package body GNAT.Bounded_Buffers is
32.
33.    --------------------
34.    -- Bounded_Buffer --
35.    --------------------
36.
37.    protected body Bounded_Buffer is
38.
39.       ------------
40.       -- Insert --
41.       ------------
42.
43.       entry Insert (Item : Element) when Count /= Capacity is
44.       begin
45.          Values (Next_In) := Item;
46.          Next_In := (Next_In mod Capacity) + 1;
47.          Count := Count + 1;
48.       end Insert;
49.
50.       ------------
51.       -- Remove --
52.       ------------
53.
54.       entry Remove (Item : out Element) when Count > 0 is
55.       begin
56.          Item := Values (Next_Out);
57.          Next_Out := (Next_Out mod Capacity) + 1;
58.          Count := Count - 1;
59.       end Remove;
60.
61.       -----------
62.       -- Empty --
63.       -----------
64.
65.       function Empty return Boolean is
66.       begin
67.          return Count = 0;
68.       end Empty;
69.
70.       ----------
71.       -- Full --
72.       ----------
73.
74.       function Full return Boolean is
75.       begin
76.          return Count = Capacity;
```

```
77.       end Full;
78.
79.       ------------
80.       -- Extent --
81.       ------------
82.
83.       function Extent return Natural is
84.       begin
85.          return Count;
86.       end Extent;
87.
88.    end Bounded_Buffer;
89.
90. end GNAT.Bounded_Buffers;
```

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

Об авторе

Пэт Роджерс был профессионалом в области вычислительной техники с 1975 года, в основном работая над приложениями на основе микропроцессора в режиме реального времени на языках Ada, C, C ++ и других языках, включая высокопроизводительные имитаторы полета и системы контроля и сбора данных (SCADA), контролирующие опасные материалы. Впервые узнав язык программирования Ada в 1980 году, он был директором лаборатории Ada9X для совместной программы Advanced Strike Technology для ВВС США, исследователя принципов в распределённых системах и исследовательских проектов отказоустойчивости с использованием Ada для ВВС США и армии, а также помощника директора по исследованиям в NASA Software Engineering Research Center. У него есть B.S. и M.S. степени в области проектирования компьютерных систем и компьютерных наук Университета Хьюстона и доктора философии в информатике из Университета Йорка, Англия. Являясь членом старшего технического персонала AdaCore, он специализируется на поддержке разработчиков программ, которые работают в режиме реального времени и для встроенных систем, создаёт и предоставляет учебные курсы, а также является руководителем проекта и разработчиком плагина GNATbench Eclipse для Ada. Он также имеет черный пояс 3-го Дан в Тэ Квон До и является основателем клуба AdaCore «The Wicked Uncles».

Last Updated: 10/13/2017
Posted on: 5/26/2008

Обсуждение…