Gem #70: Идиома Scope Locks (Идиома блокировок области кода).

Автор: Pat Rogers , AdaCore

Краткое содержание: Gem Ada #70 - Инкапсуляция разделяемых переменных в защищённых операциях не всегда возможна. В Gem Ada #70 будет показано, как добавлять взаимные исключения в существующий последовательный код с помощью комбинации контролируемых и защищённых типов так, чтобы получившийся в результате код был надёжен и максимально неизменен.

Давайте начнём…

Подобно классической концепции “монитора”, на которой они основаны, защищённые типы предоставляют взаимоисключающий доступ к внутренним переменным. Клиенты могут получить доступ к этим переменным только косвенно, с помощью процедурного интерфейса. Этот интерфейс очень надёжен, поскольку взаимоисключающий доступ предоставляется автоматически: пользователи не могут забыть получить базовую (логическую) блокировку и не могут забыть освободить её, в том числе при возникновении исключений. Поэтому настоятельно рекомендуется инкапсулировать действия в защищённые операции.

Однако применение защищённого типа и защищённых операций не всегда возможно. Например, рассмотрим существующую последовательную программу, которая вызывает процедуры и функции, предоставляемые пакетом. Внутри пакета находятся переменные, которыми манипулируют процедуры и функции. Если несколько задач теперь будут вызывать эти подпрограммы, переменные уровня пакета будут зависеть от условий гонки, поскольку они (косвенно) совместно используются вызывающими задачами. Перемещение процедур и функций в защищённый объект обеспечит необходимое взаимное исключение, но потребует изменений, как для пакета, так и для вызывающих объектов. Кроме того, существующие процедуры и функции могут выполнять потенциально блокирующие операции, такие как ввод-вывод, которые запрещены в защищённых операциях.

В таком случае программист должен вернуться к ручному получению и освобождению явной блокировки. В результате получается, по сути, использование семафоров, низкоуровневый и явно гораздо менее надёжный подход. Например, чтобы обеспечить последовательное выполнение экспортируемых операций, можно объявить блокировку на уровне пакета, как показано ниже, и каждая операция получает и освобождает её:

```
...
package body P is

   Mutex : Mutual_Exclusion;

   State : Integer := 0;

   procedure Operation_1 is
   begin
      Mutex.Seize;
      State := State + 1;  -- for example...
      Put_Line ("State is now" & State'Img);
      Mutex.Release;
   exception
      when others =>
         Mutex.Release;
         raise;
   end Operation_1;

   procedure Operation_2 is
```

```
   begin
      Mutex.Seize;
      State := State - 1;  -- for example...
      Put_Line ("State is now" & State'Img);
      Mutex.Release;
   exception
      when others =>
         Mutex.Release;
         raise;
   end Operation_2;

end P;
```

Тип Mutual_Exclusion фактически является подтипом абстракции двоичного семафора, доступной пользователям через пакет GNAT.Semaphores :

```
  subtype Mutual_Exclusion is Binary_Semaphore
    (Initially_Available => True,
     Ceiling             => Default_Ceiling);
```

См. пакет GNAT.Semaphores для деталей. Можно предположить, что это защищённый Тип с классической семантикой семафоров. Мы определяем подтип, чтобы гарантировать, что все такие объекты изначально доступны, как требуется при предоставлении взаимного исключения.

Хотя мы не можем устранить необходимость в этой блокировке, мы можем сделать код более надёжным, автоматически получая и освобождая его с помощью объекта контролируемого типа. Инициализация автоматически получает блокировку, а завершение автоматически освобождает её, в том числе при возникновении исключения и при прерывании задачи. (Программисты на C++ могут быть знакомы с этим методом под названием "Resource Acquisition Is Initialization – получение ресурсов-инициализация" (RAII).)

Идея состоит в том, чтобы определить ограниченный контролируемый тип, который ссылается на общую блокировку с помощью дискриминанта. Объекты типа затем объявляются в процедурах и функциях с дискриминантным значением, обозначающим общую блокировку, объявленную в пакете. Такой тип называется “ Scope_Lock – блокировкой области", так как для получения указанной блокировки достаточно разработки декларативного региона – области Scope, чтобы получить блокировку на которую ссылаются. Последовательность инструкций подпрограммы не будет выполняться до тех пор, пока не будет получена Блокировка, независимо от того, сколько времени это займёт. Когда процедура или функция выполнена, по любой причине, завершение снимет блокировку. Таким образом, полученный пользовательский код практически не изменяется от исходного последовательного кода:

```
...
package body P is

   Mutex : aliased Mutual_Exclusion;

   State : Integer := 0;

   procedure Operation_1 is
      S : Scope_Lock (Mutex'Access);
   begin
      State := State + 1;  -- for example...
      Put_Line ("State is now" & State'Img);
   end Operation_1;

   procedure Operation_2 is
      S : Scope_Lock (Mutex'Access);
   begin
```

```
      State := State - 1;  -- for example...
      Put_Line ("State is now" & State'Img);
   end Operation_2;

end P;
```

Чтобы определить тип Scope_Lock, мы объявляем его дискриминантом, обозначающим объект Mutual_Exclusion:

```
  type Scope_Lock (Lock : access Mutual_Exclusion) is
     tagged limited private;
```

В закрытой части Тип полностью объявлен как контролируемый Тип, производный от Ada.Finalization.Limited_Controlled, как показано ниже. Мы скрываем тот факт, что Тип будет контролироваться, потому что Initialize и Finalize никогда не предназначены для вызова вручную.

```
  type Scope_Lock (Lock : access Mutual_Exclusion) is
      new Ada.Finalization.Limited_Controlled with null record;

   overriding procedure Initialize (This : in out Scope_Lock);
   overriding procedure Finalize   (This : in out Scope_Lock);
```

Каждая переопределённая процедура просто ссылается на объект семафора, обозначенный формальным дискриминантом параметров:

```
  procedure Initialize (This : in out Scope_Lock) is
  begin
    This.Lock.Seize;
  end Initialize;

  procedure Finalize (This : in out Scope_Lock) is
  begin
    This.Lock.Release;
  end Finalize;
```

Таким образом, как вы можете видеть, комбинируя контролируемые типы с защищёнными типами, можно сделать простую работу по предоставлению взаимоисключающего доступа, когда защищаемый объект не является опцией. Используя преимущества автоматических вызовов для инициализации и завершения, полученный пользовательский код является гораздо более надёжным и требует очень небольших изменений.

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

Пэт Роджерс (Pat Rogers) 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 полёта и системы контроля и сбора данных (SCADA), контролирующие опасные материалы. Впервые узнав Аду 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 в режиме реального времени / 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: 9/21/2009

Обсуждение…

5 ответов на " Gem #70: Идиома блокировок области”

1. 23 сентября 2009 года в 18: 41

Jeff сказал:

Смотрите PragmARC.Safe_Semaphore_Handler, выпущенный 2000 мая 01, для реализации этого (в Ada 95).

http://pragmada.x10hosting.com/

2. 23 сентября 2009 года в 8: 06

Pat Rogers сказал:

Да, та же идея.

Ваш тест на несколько вызовов для завершения является важным улучшением, которое я добавлю в код gem. Спасибо за комментарий!

3. 30 сентября 2009 года в 19: 55

Pat Rogers сказал:

Jeff,

Я слишком рано изменил код. Поскольку Тип ограничен, будет только один вызов компилятора для Finalize, и поскольку он не управляется явно, клиенты не могут вызвать Finalize сами. Следовательно нет никакой потребности в защите против многократных вызовов в конце концов.

Еще раз спасибо за комментарий!

4. 1 октября 2009 года в 15: 26

Christoph Grein сказал:

Pat,

есть ли резон, почему вы делаете Scope_Lock видимым тегом?

5. 2 октября 2009, в 6:10 утра

Pat Rogers сказал:

Christoph,

В данном конкретном случае нет необходимости делать его заметным. Просто дело привычки.

Спасибо за вопрос!