Gem #41: Проверка доступности (Часть II: Ada2005)

Автор: Ramón Fernández, AdaCore

Краткое содержание: Gem Ada #41 - В Ada 2005 представлены несколько улучшений в отношении типов ссылки на объект (`access`), которые упрощают задачу разработчика и делают язык более гибким. Но большая сила означает большую ответственность, поэтому для того, чтобы новые функции не создавали недействительные ссылки, возможности проверок доступности также были расширены.

Давайте начнем…

Ada 2005 позволяет использовать анонимные типы доступа в более общем виде, добавляя значительные возможности языка объектно-ориентированного программирования. Правила доступности соответственно были расширены для обеспечения безопасности путём предотвращения возможности оборванных ссылок. Новые правила были разработаны с учётом гибкости программирования, а также позволяют компилятору применять проверки статически.

Уровни доступности в новых контекстах для типов анонимного доступа обычно определяются областью, в которой они объявлены. Это позволяет выполнять проверки специальных возможностей во время компиляции.

Другое правило, допускающее статические проверки доступности, относится к производным типам: производный Тип не создаёт новый уровень доступности для производного типа, а просто берет уровень доступности для родительского типа:

```
procedure Example_1 is
   type Node is record
      N : access Integer;
   end record;
   List : Node

   procedure P is
      type Other_Node is new Node;
   begin
      declare
         L : aliased Integer := 1;
         Data : Other_Node := Other_Node'(N => L'Access);
         --  L'Access is illegal!
      begin
         List := Node (Data);
      end;
   end P;

begin
   P;
end Example_1;
```

В приведённом выше примере нам не нужно беспокоиться о дорогостоящих проверках времени выполнения при присвоении или возврате объекта типа `Other_Node`; мы знаем, что он имеет тот же уровень доступности, что и тип `Node`, что делает атрибут `Access` незаконным. Если это не было предотвращено, после возвращения из `P`, `List.N` будет недействительной повисшей ссылкой.

Ada 2005 также позволяет функциям возвращать объекты анонимных типов доступа. В этом случае уровень доступности объекта статически определяется областью действия объявления функции. Рассмотрим следующий пример:

```
procedure Example_2 is
   type Rec is record
```

```
      V : access Integer;
      ...
   end record;

   Global : aliased Integer := 1;

   function F1 (X : Boolean) return Rec is
      Local : aliased Integer := 2;

      --  Nested function returns anonymous access values
      --  with different nesting depths

      function F2 (Y : Boolean) return Access Integer is
      begin
         if Y then
            return Global'Access;
         else
            return Local'Access;
         end if;
      end F2;

   begin
      return (V => F2 (X), ...); -- Illegal
   end F1;

   Data : Rec;
begin
   Data := F1 (True);
end Example_2;
```

В этом примере, применяя вышеупомянутое правило, компилятор статически определяет, что этот уровень доступности является областью, где объявлен `F2`, который глубже, чем уровень доступности `Rec`. Таким образом, несмотря на то, что вызов `F1 (True)` предоставляет допустимое значение для `V`, код является незаконным. Ограничение доступности является консервативным, чтобы правила были простыми, и чтобы компилятор не был обязан выполнять анализ потока данных для определения законности (не говоря уже о том, что в целом законность была бы неразрешимой).

Новые правила также учитывают дискриминаторы анонимного типа доступа (которые технически называются дискриминантами доступа). В Ada 2005 доступ к дискриминаторам теперь разрешён для не лимитированных типов. Следовательно, необходимо запретить значения по умолчанию для дискриминантов доступа не лимитированных типов. Таким образом, следующее заявление является незаконным:

```
Default : aliased Integer := ...
type Rec (D : access Integer := Default'Access) is record
   ...
```

Это ограничение необходимо для предотвращения создания дискриминантом оборванной ссылки из-за присвоения объекта `record`; оно гарантирует, что объект и дискриминант связаны друг с другом в течение их времени существования.

При использовании типов с дискриминантами доступа с распределителями – средствами выделения (allocators) и инструкциями `return` необходимо проявлять особую осторожность. Правила доступности требуют, чтобы компилятор выполнял статические проверки при создании или возврате новых объектов, содержащих дискриминанты доступа. Рассмотрим следующий пример:

```
procedure Example_3 is
   type Node (D : access Integer) is record
      V : Integer;
   end record;
   type Ptr is access all Node;
```

```
   Global_Value : aliased Integer := 1;
   Other_Data   : Integer := 2;

   procedure P is
      Local : aliased Integer := 3;
      R1 : Ptr;
      R2 : Ptr;
   begin
      R1 := new Node'(D => Global_Value'Access, V => Other_Data);
      --  This is legal

      R2 := new Node'(D => Local_Value'Access, V => Other_Data);
      --  This is illegal
   end P;
begin
   null;
end Example_3;
```

Средство выделения для `R1` законно, так как уровень доступности `Global'Access` совпадает с уровнем доступности `D`. Однако, средство выделения для `R2` недопустимо, потому что уровень доступности `Local'Access` более глубок, чем уровень доступности `D`, и присваивающий `R2` объекту вне `P` мог привести к повисшей ссылке.

Таким образом, эти правила запрещают создание объекта в пуле хранения данных, который содержит дискриминант доступа, указывающий на некоторую область памяти, будят это или часть стека или некоторого другого пула хранения данных, с более коротким временем жизни, таким образом, препятствуя тому, чтобы дискриминант указал на несуществующий объект.

Связанный исходный код из части 1

Attached Files отсутствуют

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

Об авторе

Рамон Фернандес - старший инженер-программист AdaCore. С момента прихода в компанию в 2001 году он участвовал в самых разных областях, включая поддержку клиентов, GNAT Pro для разработчиков и разработчиков, системы реального времени, обеспечение качества, встроенные системы, производственную инфраструктуру и управление ИТ. Рамон имеет степень магистра компьютерных наук в Нью-Йоркском университете и степень электроники в области телекоммуникаций от ETSIT-UPM (технический университет Мадрида, Испания).

Last Updated: 10/13/2017
Posted on: 6/30/2008

Обсуждение…