Gem #123: Неявное разыменование в Ada 2012

Автор: Christoph Grein

Краткое содержание: В Gem #107 был описан метод чтения для создания безопасных ссылок на объекты контейнеров. В примере описывался указатель с подсчитанными ссылками, но подобные методы можно определять и для других контейнеров.

Давайте начнём…

В Gem # 107 мы представили средство доступа для безопасной ссылки на объекты, хранящиеся в контейнере. Пример касается указателя с подсчетом ссылок, но такие методы доступа могут быть определены для любого типа контейнера.

Преимущество использования средств доступа, а не простых типов доступа, состоит в том, что первое не может использоваться для освобождения назначенного объекта. Однако безопасность всегда сопряжена с издержками, в данном случае в форме неудобного синтаксиса. В этом Gem мы покажем, как некоторые функции Ada 2012 можно использовать для упрощения синтаксиса.

Рассмотрим прототип контейнера в качестве примера:

```
generic
   type Element is private;
   type Key     is private;
   with function Key_of (E: Element) return Key;
package Containers is

   type Container is private;

   type Accessor (Data: not null access Element) is limited private;

   procedure Put (C: in out Container; E: in Element);

   function  Get (C: Container; K: Key) return Accessor;

private

   ... implementation not shown
end Containers;
```

Контейнер содержит элементы, которые можно получить с помощью некоторого ключа. Как элементы хранятся и извлекаются здесь не представляет интереса. Важно то, что Get предоставляет прямой доступ к хранимому элементу (другими словами, Get не возвращает копию). Это очень важно, если у вас большой объект и вы хотите обновить только компонент:

```
Get (Cont, My_Key).Data.Component := Some_Value;
```

Обратите внимание, что если дискриминант должен быть определен как

```
not null access constant Element
```

, тогда средство доступа позволило бы только доступ для чтения. Кроме того, пустое исключение гарантирует, что средство доступа будет всегда ссылаться на объект.

Этот синтаксис является довольно подробным, но Ada, которую 2012 обеспечивает новой возможности, которая помогает упростить его, а именно, аспект Implicit_Dereference.

Примечание стороны: Ada 2012 добавил общий механизм, названный спецификацией аспекта, которая позволяет определять различные характеристики объявлений, названных аспектами, как часть самого объявления. Например, атрибуты представления могут теперь быть определены при помощи спецификации аспекта, а не отдельного определения атрибута.

Вот то, как аспект Implicit_Deference был бы определен для нашего типа Accessor type:

```
type Accessor (Data: not null access Element) is limited private
   with Implicit_Dereference => Data;
```

Тип, определенный с этим аспектом, называют ссылочным типом, и объект такого типа является ссылочным объектом. Использование этого аспекта позволяет нам уменьшать оператор до:

```
Get (Cont, My_Key).Component := Some_Value;
```

Обратите внимание, что вызов `Get (Cont, My_Key)` перегружен: его результат может быть интерпретирован как значение метода доступа или сам объект доступа, и компилятор разрешает его на основе контекста вызова. (Именно по этой причине аспект `Implicit_Dereference` нельзя использовать для указателя с подсчетом ссылок в Gem #107, где требуется преобразование типов, так как аргумент преобразования типов должен быть разрешен независимо от контекста.)

Вы можете возразить, что это не является существенным упрощением. Однако на этом история не заканчивается. Нас интересует не столько как получить значение метода доступа (результат функции `Get`), сколько как добраться до самих элементов. Бывает, что мы можем избежать вызова `Get` с помощью другого аспекта, называемого `Variable_Indexing`, который применяется к контейнерному типу:

```
type Container is tagged private
   with Variable_Indexing => Get;
```

Тип результата функции `Variable_Indexing` должен быть ссылочным типом. Стоит отметить, что имя, указанное в спецификации аспекта, может означать что-то объявленное позже. В этом случае это прямая ссылка на `Get`, которая объявляется после типа.

Кроме того, необходимо пометить тип, к которому применяется атрибут `Variable_Indexing`. Будучи помеченным типом, это позволяет `Object.Operation` операций, приводящие к:

```
Cont.Get (My_Key).Component := Some_Value;
```

Учитывая `Variable_Indexing` аспект, который определяет `Get`, это теперь может быть далее сведено к просто:

```
Cont (My_Key).Component := Some_Value;
```

Фактически, мы получаем прямой доступ к элементам контейнера, как если бы контейнер был своего рода массивом, индексированным ключом. Фактически, индексированное имя можно использовать только в контексте, требующем переменной типа элемента, например оператора присваивания:

```
Cont (My_Key) := Some_Element_Value;
```

Таким образом, объединяя новые аспекты `Implicit_Dereference` и `Variable_Indexing`, мы получаем сжатый и гораздо более читаемый синтаксис для манипулирования элементами контейнера.

Кстати, есть также сопутствующий аспект `Variable_Indexing`, называемый `Constant_Indexing`, который может использоваться для предоставления доступа только для чтения к значениям элементов. В этом случае связанная функция не обязана возвращать ссылочный тип, поскольку все функции возвращают постоянный результат.

Связанный со статьёй текст программы

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

Об авторе

Сведений об авторе нет

Last Updated: 10/13/2017
Posted on: 4/17/2012

Обсуждение…