Gem #57: Диспетчеризация вызовов между Ada и Java.

Автор: Quentin Ochem, AdaCore

Краткое содержание: Gem Ada #57 - в предыдущем Gem #56 было показано, как создавать вызовы Java в Ada с помощью обратных вызовов и функции ada2java. Следующий шаг - это механизм межъязыковой диспетчеризации для поддержки расширения тегированного типа Ada в Java, что позволяет осуществлять взаимные вызовы диспетчеризации.

Давайте начнем…

Рассмотрим следующий API:

```
package API is
   type A_Tagged_Type is tagged null record;

   procedure Print_Me (V : A_Tagged_Type; Me : String);

   type An_Ada_Child is new A_Tagged_Type with null record;

   procedure Print_Me (V : An_Ada_Child; Me : String);

   procedure Call_Print_Me (Str : String; Val : A_Tagged_Type'Class);
end API;

package body API is
   procedure Print_Me (V : A_Tagged_Type; Me : String) is
   begin
      Put_Line ("FROM A TAGGED TYPE: " & Me);
   end Print_Me;

   procedure Print_Me (V : An_Ada_Child; Me : String) is
   begin
      Put_Line ("FROM AN ADA CHILD: " & Me);
   end Print_Me;

   procedure Call_Print_Me (Str : String; Val : A_Tagged_Type'Class) is
   begin
      Print_Me (Val, Str);
   end Call_Print_Me;
end API;
```

Обратите внимание на то, что вызов в Ada `Call_Print_Me` диспетчеризирует, поэтому если я пишу в Ada:

```
V1 : A_Tagged_Type;
V2 : An_Ada_Child;
Call_Print_Me ("V1", V1);
Call_Print_Me ("V2", V2);
```

Это вызовет `Call_Print_Me` фактического объекта, так что Выходные данные будут:

```
FROM A TAGGED TYPE: V1
FROM AN ADA CHILD: V2
```

Генерация обязательного кода

Как обычно, создание всей привязки выполняется с помощью простой команды:

```
ada2java api.ads -b test -o ada -c java -L my_lib
```

Написание кода Java

Теперь у нас есть два класса: `test.API.A_Tagged_Type` и `test.API.An_Ada_Child`, оба из которых являются регулярными незавершенными java-классами, а `An_Ada_Child` - явно из `A_Tagged_Type`. Поэтому мы можем написать пример, аналогичный тому, что мы делали в Ada:

```
import test.API_Package;
import test.API.A_Tagged_Type;
import test.API.An_Ada_Child;
import test.Standard.AdaString;

public class My_Main {

public static void main (String [] argv) {
   A_Tagged_Type v1 = new A_Tagged_Type ();
   An_Ada_Child v2 = new An_Ada_Child ();
   API_Package.Call_Print_Me (new AdaString ("V1"), v1);
   API_Package.Call_Print_Me (new AdaString ("V2"), v2);
}
```

который будет иметь точно такой же эффект. Но мы можем пойти ещё дальше; можно получить класс `A_Tagged_Type` в Java и переопределить его примитив. Давайте сделаем это в классе, вложенном в `My_Main`, например:

```
static class A_Java_Child extends A_Tagged_Type {
   public void Print_Me (AdaString Me) {
      System.out.println ("FROM A JAVA CHILD" + Me);
   }
}
```

Обратите внимание, что здесь примитив диспетчеризации Ada теперь является функцией-членом Java. Первый параметр подпрограммы Ada, управляющий операнд "V“, был привязан к неявному параметру ”`this`" Java, так, чтобы член заканчивал тем, что имел только один явный параметр вместо двух. Кстати, такая конструкция была бы невозможна, если бы первый параметр примитива Ada не контролировал, и ada2java предупредил бы нас об этом.

Теперь мы можем использовать экземпляры этого объекта так же, как экземпляры любого из его родителей, например:

```
public static void main (String [] argv) {
   A_Tagged_Type v1 = new A_Tagged_Type ();
   An_Ada_Child v2 = new An_Ada_Child ();
   A_Tagged_Type v3 = new A_Java_Child ();
   API_Package.Call_Print_Me (new AdaString ("V1"), v1);
   API_Package.Call_Print_Me (new AdaString ("V2"), v2);
   API_Package.Call_Print_Me (new AdaString ("V3"), v3);
}
```

И точно так же, как видно из этого примера, мы ввели в действие диспетчеризирующий межъязыковым образом вызов между Ada и Java. На стороне Ada код, сгенерированный обязательным генератором и AJIS run-time, будет в состоянии обнаружить, что этому последнему объекту расширяли тип в Java, и что вызов диспетчеризации должен быть разрешён, используя объект Java.

Компиляция и запуск

Как и в предыдущих версиях «Ada/Java interfacing» Gems #55 и #56, команды compile и run в Linux разбиваются на следующую последовательность:

```
gprbuild -p -P ada/my_lib.gpr
LD_LIBRARY_PATH=`pwd`/ada/lib/:$LD_LIBRARY_PATH
```

```
export LD_LIBRARY_PATH
CLASSPATH=`pwd`:`pwd`/java:<your AJIS
installation>/lib/ajis.jar:$CLASSPATH
javac My_Main.java
java My_Main
```

которая отображает на экране результат работы программы `My_Main`:

```
FROM A TAGGED TYPE: V1
FROM AN ADA CHILD: V2
FROM A JAVA CHILD: V3
```

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

Quentin Ochem имеет опыт разработки программного обеспечения, специализирующийся на разработке программного обеспечения для критически важных приложений. Он имеет более чем 10-летний опыт разработки Ada. Сегодня он работает техническим менеджером AdaCore по проектам, связанным с авионикой, железнодорожной, космической и оборонной отраслями. Он также обучает стандарту DO-178B авионики в университете EPITA в Париже.

Last Updated: 10/13/2017
Posted on: 1/26/2009

Обсуждение…