Gem #56: Создание вызовов Java в Ada с помощью GNAT-AJIS

Автор: Quentin Ochem, AdaCore

Краткое содержание: Gem Ada #56 - В предыдущем Gem #55 была представлена функция ada2java, связывающее спецификирования Ada и Java для поддержки вызовов Ada в Java. Несмотря на то, что ada2java не поддерживает пакеты сопряжений Ada с спецификированиями Java, его можно использовать для поддержки вызовов Java в Ada. В данном Gem будет рассмотрена такая возможность на примере обратных вызовов (в контексте Ada - вызовов доступа к подпрограммам).

Давайте начнем…

Рассмотрим следующий API:

```
package API is

   type A_Callback is access procedure (Str : String; Val : Integer);

   procedure Process (Str : String; C : A_Callback);

end API;

package body API is
   procedure Process (Str : String; C : A_Callback) is
   begin
      C.all (Str, 1);
      C.all (Str, 2);
   end Process;
end API;
```

В этом примере `A_Callback` вызывается дважды из `Process`. Наша цель здесь состоит в том, чтобы эти вызовы вызывали Java-код.

Создание кода привязки

Как и в предыдущем Gem, генерация кода привязки выполняется с помощью консольной утилиты ada2java инструмента GNAT-AJIS (используется Java Native Interface (JNI), с помощью которого осуществляется взаимодействие среды машинного кода и JVM):

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

Подробности о значении аргументов команды можно найти в предыдущем Gem #55 или в документации ada2java.

Написание кода Java

Посмотрев на сгенерированный код, вы увидите, что функция `Process` была связана следующим образом:

```
public final class API_Package {
   […]
   static public void Process (test.Standard.AdaString Str, test.API.A_Callback C){
   […]
}
```

с `test.API.A_Callback`:

```
public abstract class A_Callback {
   […]
   abstract public void A_Callback_Body (test.Standard.AdaString Str, int Val);
```

```
   […]
}
```

Теперь все, что необходимо для реализации Ada access-to-subprogram в Java, состоит в том, чтобы из `A_Callback` создать конкретный класс, реализовать его элемент `A_Callback_Body`, а затем предоставить экземпляр этого объекта для функции `Process`.

Например:

```
import test.API.A_Callback;
import test.API.API_Package;
import test.Standard.AdaString;

public class My_Main {
   static class My_Callback extends A_Callback {
      public void A_Callback_Body (AdaString Str, int Val) {
         System.out.println ("Call " + Val + " of " + Str);
      }
   }
   public static void main (String [] argv) {
      API_Package.Process (new AdaString ("Hello"), new My_Callback ());
   }
}
```

И так легко, вы установили обратный вызов, с Java, вызывающим Ada, а затем переходом на Java!

Компиляция и запуск

Как и в предыдущем gem, мы сгенерировали файл проекта сборки с параметром « `-L` ». Таким образом, следуя этим шагам, на Linux, вам нужно сделать:

```
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
```

Теперь, если вы попытаетесь запустить программу, она не будет работать так, как ожидалось. Оказывается, что исключение возникает в точке вызова `Process`, на стороне Java. Давайте рассмотрим это подробнее.

Решение проблемы “escaped” при запуске программы

Проблема заключается в том, что невозможно хранить объекты класса `A_Callback`, созданные из Java – они не являются фактическими значениями подпрограммы доступа на уровне библиотеки, а являются объектами Java. Возможно, вы захотите взглянуть на сгенерированный код ADA glue, если вам нужны дополнительные сведения об этом, но важно то, что такие объекты не могут использоваться машинным кодом после вызова `Process`, поэтому они не должны храниться. Мы скажем, что им нельзя “escaped” (уцелеть).

К сожалению, мы не можем гарантировать, что объекты действительно не уцелели, поэтому AJIS run-time просто отказывается передавать такие объекты, чтобы Разработчик не мог их оставить уцелевшими избежать их уничтожение по ошибке. В данном случае мы просто делаем два вызова. Ни один объект не будет сохранен, и вызовы для обработки совершенно безопасны. Мы можем дать эту информацию генератору привязки, используя одну из аннотаций AJIS:

```
with AJIS.Annotations;

package API is
   type A_Callback is access procedure (Str : String; Val : Integer);

   procedure Process (Str : String; C : A_Callback);
   pragma Annotate (AJIS, Assume_Escaped, False, Process, "C");
```

Директива pragma указывает, что параметр `C` процесса не следует считать потенциально экранированным – другими словами, Разработчик Ada берет на себя ответственность за то, чтобы не позволить ему уйти. Среда выполнения AJIS затем позволит ему передавать нескрываемые объекты в качестве параметров.

Обратите внимание, что теперь у нас есть ссылка на время выполнения AJIS (AJIS run-time) в наших рабочих источниках кода. Самый простой способ ссылаться на эту среду выполнения - использовать файл проекта для управления источниками, например, с помощью следующего файла проекта:

```
with "ajis";
project API is
end API;
```

и затем связывание может быть регенерировано:

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

После перекомпиляции Ada и источников Java, мы повторно запускаем JAVA-приложение, и вызовы от Ada к Java теперь хорошо работают!

В следующем Gem #57 мы посмотрим на то, как сделать межъязыковую диспетчеризацию, расширив Ada теговый тип в Java.

Связанный со статьёй текст программы

Attached Files отсутствуют

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: 1/12/2009

Обсуждение…