Gem #58: Обработка исключений Ada/Java

Автор: Quentin Ochem, AdaCore

Краткое содержание: Gem Ada #58 - Ada и Java сильно зависимы от исключений. Модель данных Ada основывается на том, что данные проверяются в процессе исполнения, после чего она отображает различные исключения, например, `Constraint_Error` в случае ошибки, связанной с ограничениями. В Java также производятся проверки, в ходе которых может быть обнаружено множество исключений, чаще всего - проверки преобразований типов и ответ null при запросе значения переменной объекта по указателю. Поэтому очень важны исключения, развитые в обоих языках, которые даже потенциально можно обнаружить/обработать без знания того, в каком именно языке они образовались.

Давайте начнем…

Рассмотрим следующий API:

```
with AJIS.Annotations; use AJIS.Annotations;

package API is

   function Compute (I : Integer) return Integer;

   type An_Operation is access function (I : Integer) return Integer;

   function Indirect_Compute (Data : Integer; Operation : An_Operation)
     return Integer;
   pragma Annotate (AJIS, Assume_Escaped, False, Indirect_Compute,
"Operation");

end API;
```

с последующей реализацией:

```
package body API is

   function Compute (I : Integer) return Integer is
      Tmp : Natural := I;
   begin
      return Tmp + 1;
   end Compute;

   function Indirect_Compute (Data : Integer; Operation : An_Operation)
     return Integer
   is
   begin
      return Operation.all (Data);
   exception
      when Constraint_Error =>
         return 0;
   end Indirect_Compute;

end API;
```

Подпрограмма Compute использует временную переменную подтипа `Natural` и вызывает `Constraint_Error`, если передаётся отрицательное значение для `I`. С другой стороны, подпрограмма `Indirect_Compute` перехватывает все `Constraint_Errors` и возвращает `0`, если это происходит.

Генерация обязательного кода

Как и в предыдущих Ada/Java interfacing Gems, генерация привязки выполняется с помощью простой команды:

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

Обратите внимание, что вам нужен файл проекта api.gpr, который определяется следующим образом:

```
with "ajis";

project API is

end API;
```

Написание кода Java

Давайте теперь немного поиграем с этим кодом. Как вы можете видеть, классы Java генерируются для исключений Ada, таких как `test.Standard.Constraint_Error`. Эти исключения заканчиваются как обычные исключения Java, которые мы можем уловить, например, в следующем коде:

```
import test.API.API_Package;
import test.Standard.Constraint_Error;

public class My_Main {

   public static void main (String [] argv) {

      try {
         int x = API_Package.Compute (-1);
      } catch (Constraint_Error e) {
      System.out.println (e.getMessage ());
      }
   }
}
```

И точно так же у нас есть исключение, поднятое из Ada, преобразованное в исключение Java и, наконец, попавшее в блок Java. Но давайте сделаем что-то более амбициозное. Возьмем исключение из Ada, получим ли это исключение через кадры Java, а затем вернёмся в Ada:

```
import test.API.API_Package;
import test.API.An_Operation;
import test.Standard.Constraint_Error;

public class My_Main {

   public static void main (String [] argv) {
      int x = API_Package.Indirect_Compute (-1,
                  new An_Operation () {
                        public int An_Operation_Body (int I) {
                              return API_Package.Compute (I);
                        }
                  }
```

```
                    );
    }
}
```

Итак, здесь у нас есть обратный вызов, вызванный из Ada, реализованный на Java, который вызывает процедуру Ada `Compute`. Давайте посмотрим, что происходит в деталях. Во-первых, `Indirect_Compute` вызывается Java, переходит в Ada, который затем вызывает подпрограмму, переданную как параметр доступа, приводящую к вызову Java. Затем эта функция Java вызывает операцию Ada `Compute`, которая вызовет исключение из-за прохождения `-1`. Таким образом, в точке исключения стек вызовов выглядит следующим образом:

```
[1] Java : main
[2] Ada : Indirect_Compute
[3] Java : An_Operation_Body
[4] Ada : Compute <- the exception is raised here
```

Исключение сначала идёт от [4] до [3], как в предыдущем примере. В этом случае, однако, это не поймано, таким образом, это распространено к вызывающей стороне, [2], который, оказывается, код Ada снова. К счастью, код связующего звена, сгенерированный между [2] и [3], в состоянии перевести исключение назад от Java до исходного исключения Ada и продолжать распространение. В этом случае есть обработчик исключений в [2], который обрабатывает `Constraint_Error` и поймает тот, первоначально повышенный в [4].

Компиляция и запуск

Как и в предыдущих версиях «Ada/Java interfacing» Gems #55, #56 и #57, 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
```

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: 2/9/2009

Обсуждение…