Gem #142: Исключительный друг исключений

Автор: Emmanuel Briot — AdaCore

Краткое содержание: Компилятор GNAT известен благодаря качеству генерируемых им сообщений об ошибках. Это относится и к сообщениям, связанным с исключениями. В данном Gem будет продемонстрировано, как можно извлечь из них пользу.

Давайте начнём…

Стандартная библиотека времени выполнения Ada предоставляет пакет `Ada.Exceptions`. Этот пакет предоставляет ряд услуг, помогающих анализировать исключения.

Каждое исключение связано с (коротким) сообщением, которое может быть установлено кодом, который повышает (обрабатывает) исключение. В более свежих версиях Ada эти сообщения могут быть установлены очень просто, как в следующем коде:

```
Ada.Exceptions.Raise_Exception          --  Ada 95
    (Constraint_Error'Identity, "some message");

raise Constraint_Error with "some message";  --  Ada 2005
```

Новый синтаксис теперь очень удобен, и разработчикам следует рекомендовать предоставить как можно больше информации вместе с исключением. Однако длина сообщения ограничена 200 символами по умолчанию в GNAT, и сообщения длиннее этого будут усечены.

Исключения также встраивают информацию, заданную самой средой выполнения, которую можно получить, вызвав функцию `Exception_Information`. В случае GNAT эта информация может включать местоположение источника, в котором было вызвано исключение, и несимметричную трассировку. Функция `Exception_Information` также отображает `Exception_Message`. Вот типичный обработчик исключений, который ловит все непредвиденные исключения в приложении:

```
pragma Ada_05;
with Ada.Exceptions;
with Ada.Text_IO;   use Ada.Text_IO;

procedure Main is

    procedure Nested is
    begin
        raise Constraint_Error with "some message";
    end Nested;

begin
    Nested;

exception
    when E : others =>
        Put_Line (Ada.Exceptions.Exception_Information (E));
end Main;
```

Давайте теперь скомпилируем это приложение без специальной опции командной строки.

```
> gnatmake main.adb
> ./main
```

```
Exception name: CONSTRAINT_ERROR
Message: some message
```

Это не очень информативно. Чтобы получить дополнительную информацию, нам нужно перезапустить программу в отладчике. Чтобы сделать сеанс более интересным, мы должны добавить отладочную информацию в исполняемый файл, что означает использование переключателя `-g` в команде `gnatmake`.

Сеанс будет выглядеть следующим образом (опуская некоторые выходные данные из отладчика):

```
  > rm *.o      # Cleanup previous compilation
  > gnatmake -g main.adb
  > gdb ./main
  (gdb)  catch exception
  (gdb)  run
Catchpoint 1, CONSTRAINT_ERROR at 0x0000000000402860 in main.nested () at main.adb:8
8               raise Constraint_Error with "some message";

  (gdb) bt
#0  <__gnat_debug_raise_exception> (e=0x62ec40 <constraint_error>) at s-excdeb.adb:43
#1  0x000000000040426f in ada.exceptions.complete_occurrence (x=x@entry=0x637050)
    at a-except.adb:934
#2  0x000000000040427b in ada.exceptions.complete_and_propagate_occurrence (
    x=x@entry=0x637050) at a-except.adb:943
#3  0x00000000004042d0 in <__gnat_raise_exception> (e=0x62ec40 <constraint_error>,
    message=...) at a-except.adb:982
#4  0x0000000000402860 in main.nested ()
#5  0x000000000040287c in main ()
```

И теперь мы точно знаем, где было создано исключение.

Но на самом деле, мы могли бы иметь эту информацию непосредственно при выполнении приложения.

Для этого нам нужно связать приложение с коммутатором `-E`, который говорит редактору связей хранить исключение tracebacks во вхождениях исключительных ситуаций. Давайте перекомпилируем и давайте повторно выполним приложение.

```
  > rm *.o    # Cleanup previous compilation
  > gnatmake -g main.adb -bargs -E
  > ./main

Exception name: CONSTRAINT_ERROR
Message: some message
Call stack traceback locations:
0x10b7e24d1 0x10b7e24ee 0x10b7e2472
```

Трассировка, как есть, не очень полезна. Теперь нам нужно использовать другой инструмент, связанный с GNAT, который называется `addr2line`. Вот пример его использования:

```
  > addr2line -e main --functions --demangle 0x10b7e24d1 0x10b7e24ee
0x10b7e2472
/path/main.adb:8
_ada_main
/path/main.adb:12
main
/path/b~main.adb:240
```

На этот раз у нас есть символический backtrace, который показывает информацию, похожую на то, что мы получили в отладчике.

Для пользователей на компьютерах OSX `addr2line` не существует. Однако на этих машинах, существует эквивалентное решение. Вам нужно связать приложение с дополнительным коммутатором, а затем использовать инструмент `atos`, как в:

```
   > rm *.o
   > gnatmake -g main.adb -bargs -E -largs -Wl,-no_pie
   > ./main

Exception name: CONSTRAINT_ERROR
Message: some message
Call stack traceback locations:
0x1000014d1 0x1000014ee 0x100001472
   > atos -o main 0x1000014d1 0x1000014ee 0x100001472
main__nested.2550 (in main) (main.adb:8)
_ada_main (in main) (main.adb:12)
main (in main) + 90
```

Теперь мы обсудим относительно новый параметр компилятора, а именно `-gnateE`. При использовании этот переключатель генерирует дополнительную информацию в сообщениях об исключениях.

Давайте изменим нашу тестовую программу так:

```
pragma Ada_05;
with Ada.Exceptions;
with Ada.Text_IO;       use Ada.Text_IO;

procedure Main is

    procedure Nested (Index : Integer) is
       type T_Array is array (1 .. 2) of Integer;
       T : constant T_Array := (10, 20);
    begin
       Put_Line (T (Index)'Img);
    end Nested;

begin
    Nested (3);

exception
    when E : others =>
        Put_Line (Ada.Exceptions.Exception_Information (E));
end Main;
```

Мы скомпилируем его с дополнительным ключом и затем запустим:

```
  > gnatmake -g main.adb -gnateE -bargs -E -g -largs
  > ./main

Exception name: CONSTRAINT_ERROR
Message: main.adb:10:18 index check failed
index 3 not in 1..2
```

```
Call stack traceback locations:
0x100001429 0x1000014c7 0x1000013c2
```

Информация об исключении (traceback) такая же, как и раньше, но на этот раз сообщение об исключении устанавливается компилятором автоматически. Итак, мы знаем, что получили `Constraint_Error`, потому что в названном исходном местоположении был использован неверный индекс (строка 10 `main.adb`). Но значительным дополнением является вторая строка сообщения, в которой точно указана причина ошибки. Здесь мы хотели получить элемент с индексом 3 в массиве, чей диапазон допустимых индексов составляет от 1 до 2.

Нет необходимости в отладчике в этом случае.

Информация о столбцах в первой строке сообщения об исключении также очень полезна при работе с нулевыми указателями. Например, такая строка:

```
A := Rec1.Rec2.Rec3.Rec4.all;
```

где каждая из Rec сама является указателем, может вызвать `Constraint_Error` с сообщением "`access check failed`" («проверка доступа не удалась»). Это точно указывает на то, что один из указателей является нулевым, и, используя информацию столбца, как правило, легко определить, какой это указатель.

Связанный со статьёй текст программы

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

Об авторе

Автор: Emmanuel Briot — AdaCore

Эммануэль Брио работает в AdaCore с 1998 года. Он принимал участие в различных проектах, в частности, ориентированных на графические пользовательские интерфейсы, включая GtkAda, GPS, XML / Ada, GnatTracker и нашу внутреннюю CRM. Он получил степень инженера в Национальной школе телекоммуникаций - Брест, Франция (Ecole Nationale des Telecommunications - Brest, France).

Last Updated: 10/13/2017
Posted on: 2/25/2013

Обсуждение…