Gem #143: Возвращение к источникам – Понятие проектов

Автор: Emmanuel Briot — AdaCore

Краткое содержание: Понятие проектов было введено в технологию GNAT начиная с версии 3.15 2002 года. В дальнейших версиях его реализация была улучшена, и теперь с проектами могут взаимодействовать все инструменты GNAT. С их помощью удобно описывать организацию источников приложений и то, как с ними должны взаимодействовать различные инструменты. С ними также могут взаимодействовать пользовательские инструменты с помощью удобных API из набора компонентов GNAT, которые и описываются в данном Gem.

Давайте начнём…

Все начинается с источника.

Все большие приложения организуют свой исходный код в несколько отдельных каталогов, которые мы обычно считаем модулями. Сами исходные файлы обычно следуют соглашениям об именах, чтобы мы могли легко что-то найти. Например, традиционным расширением для файлов Ada (в GNAT) являются `.adb` и `.ads`, хотя в других технологиях используются другие расширения (например, `.1.ada`).

Многие инструменты, в частности компилятор и IDE, должны находить исходные файлы для выполнения различных действий над кодом. Однако, как только они найдут источники, им также нужно знать, как ими манипулировать. Например, компилятору может потребоваться скомпилировать определенный файл с отключенными проверками стиля, тогда как для всех остальных файлов необходимо включить проверки стиля, чтобы обеспечить согласованность стиля.

В настоящее время типичное приложение использует несколько языков, таких как Ada и C. Каждый язык имеет свою собственную схему именования и, возможно, свой собственный набор инструментов.

Сначала GNAT использовал ключи типа `-I` для указания на разные исходные каталоги и ожидал, что все исходные файлы будут использовать суффиксы `.adb` и `.ads`. Но затем мы представили файлы проектов, которые служат удобным местом для описания организации программных проектов. В отличие от `Makefile`, они являются чисто описательными и не описывают набор действий, которые нужно выполнить. Это делает их идеально подходящими для совместного использования несколькими инструментами.

Другие Gems уже говорили о различных аспектах проектов, поэтому мы не будем вдаваться в подробности здесь.

Однако может случиться так, что ваше собственное приложение сможет использовать то, что находится в файлах проекта. Эффективный их анализ сложен, так как мы продолжаем добавлять функции для поддержки различных аспектов управления источниками, и анализатор должен постоянно обновляться.

Вместо этого мы рекомендуем использовать пакет `GNATCOLL.Projects`, находящийся в коллекции компонентов GNAT, для манипулирования файлами проекта. Этот Gem представляет краткое введение в функции этого пакета.

Вот простой пример использования:

```
pragma Ada_05;
with GNATCOLL.Projects;   use GNATCOLL.Projects;
with GNATCOLL.VFS;        use GNATCOLL.VFS;   --  Gem 118
...
```

```
declare
   Tree : Project_Tree;
begin
   Tree.Load (GNATCOLL.VFS.Create ("root.gpr"));
end;
```

Между терминами часто возникает путаница. Давайте дадим несколько определений, которые используются в API GNATCOLL. Дерево проекта - это набор проектов, которые могут зависеть друг от друга. Вы можете думать о дереве как о представлении всего приложения или исходной базы. Обычно он подразделяется на модули, каждый из которых содержит один файл проекта.

В приведенном выше примере мы загрузили дерево с корнем в root.gpr. Таким образом, мы загрузили проект root.gpr, но, возможно, и проект child.gpr, от которого зависит root.gpr.

Контекст

На самом деле, приведенный выше пример часто упрощен. Проект можно настроить для нескольких сценариев (используя ключевое слово «external» в файле проекта, а затем некоторые операторы case, например, для изменения списка исходных каталогов в зависимости от переменной среды).

Точно так же ваш проект может зависеть от некоторых предустановленных проектов. Например, если вы намереваетесь использовать GNATCOLL.Projects, ваш проект, скорее всего, будет зависеть от gnatcoll.gpr. Чтобы найти эти проекты, GNATCOLL по умолчанию спросит у gnatls, где, по его мнению, находятся предопределенные проекты или каков каталог времени выполнения. Но вы также можете добавить свой собственный.

Для этого вам нужно пройти через экземпляр Project_Environment, как показано в следующем коде:

```
declare
   Env : Project_Environment_Access;
begin
   Initialize (Env);
   Env.Set_Predefined_Source_Path ((1 => Create
("/usr/local/prefix")));

   --  add a custom language
   Env.Register_Default_Language_Extension ("python", ".py", "");

   --  set up scenario variables
   Env.Change_Environment ("VARIABLE", "VALUE");

   Tree.Load (Create ("root.gpr"), Env => Env);
end;
```

На этот раз проект загружается в определенном, предварительно инициализированном контексте, что может повлиять на представление приложения.

Если вам нужно изменить сценарий в течение срока действия вашего приложения, вы должны сделать следующее:

```
Env.Change_Environment ("VARIABLE", "VALUE2");
Tree.Recompute_View;
```

который перезагружает тот же проект в другом сценарии. Теперь, например, список исходных файлов или переключателей компилятора может быть другим.

Запросы

Как только мы загрузим проекты в память, нам нужно выполнить запросы для извлечения информации.

Прежде всего, давайте найдем список всех исходных файлов в приложении.

```
pragma Ada_12;    --  convenient for iterators
...
declare
   Src : File_Array_Access :=
           Tree.Root_Project.Source_Files (Recursive => True);
begin
   for S of Src loop
      Put_Line (S.Display_Full_Name);
   end loop;
   Free (Src);
end;
```

Исходные файлы возвращаются как экземпляры `Virtual_File`. Как мы видели в Gem 118, такой объект обеспечивает удобный кэш для информации, которая в противном случае должна была бы запрашиваться с помощью системных вызовов, которые могут быть медленными в некоторых системах. Кроме того, он не предполагает, что вам понадобится полный путь, базовое имя или другая информация. Эти объекты кэшируются в дереве проекта, так что каждый раз, когда вы запрашиваете исходные файлы, возвращаются те же экземпляры (и его кэш).

Частая операция заключается в том, что у вас есть базовое имя для исходного файла (например, `a.adb`), и хотите найти его на диске. Этого можно легко достигнуть с: Исходные файлы возвращаются как экземпляры `Virtual_File`. Как мы видели в Gem 118, такой объект обеспечивает удобный кэш для информации, которая в противном случае должна была бы запрашиваться с помощью системных вызовов, которые могут быть медленными в некоторых системах. Кроме того, он не предполагает, что вам понадобится полный путь, базовое имя или другая информация. Эти объекты кэшируются в дереве проекта, так что каждый раз, когда вы запрашиваете исходные файлы, возвращаются те же экземпляры (и его кэш).

Частая операция заключается в том, что у вас есть базовое имя для исходного файла (например, `a.adb`), и хотите найти его на диске. Этого можно легко достигнуть с:

```
declare
   A_Adb : constant Virtual_File := Tree.Create ("a.adb");
begin
   Put_Line (A_Adb.Display_Full_Name);
end;
```

Опять же, эта информация кэшируется, поэтому очень быстро выдается на запрос.

Схемы именования

Как можно больше, инструменты должны быть в состоянии обработать несколько исходных языков. От исходного файла мы, поэтому должны знать его язык, который может быть сделан с:

```
   Put_Line (Tree.Info (A_Adb).Language);     --  "ada"
```

У Ada, в частности, также есть понятие модуль. Например, модуль GNATCOLL.Projects находится в исходном файле "gnatcoll-projects.ads". Отображение от одного до другого полностью описано в файле проекта и не является чем-то, что каждое приложение должно принять или перекомпилировать самостоятельно.

Из исходного файла получение название модуля выполняется с помощью:

```
Put_Line (Tree.Info (A_Adb).Unit_Name);   --  "A"
Put_Line (Tree.Info (A_Adb).Unit_Part);   --  Unit_Body
```

Из устройства поиск исходного файла выполняется с помощью:

```
Put_Line (Tree.Root_Project ("A", Unit_Body, "Ada"));  --  "a.adb"
```

Атрибуты

Информация в файле проекта организована в пакеты (обычно по одному для каждого инструмента, такого как компилятор, binder, IDE,...), а затем в атрибуты. Пользователи могут добавлять свои собственные пакеты, поэтому вы можете решить, что конфигурация вашего собственного инструмента должна входить в пакет My_Tool и какие атрибуты можно использовать для этой конфигурации. Однако не следует добавлять новые атрибуты в предопределенные пакеты, так как анализатор проекта будет жаловаться в противном случае, чтобы избежать возможных будущих столкновений имен.

Типичным атрибутом является Switches, который задает параметры командной строки для передачи инструменту. Таким образом, файл проекта пользователя может содержать:

```
project Root is
   package My_Tool is
      for Switches ("Ada") use ("-a", "-b");
   end My_Tool;
end Root;
```

Из программы на язык Ada вы можете получить значение этого атрибута со следующим фрагментом кода.

```
pragma Ada_12;
with GNAT.Strings;   use GNAT.Strings;
...
declare
 My_Tool_Switches : constant Attribute_Pkg_List :=
  Build (Package_Name => "My_Tool", Attribute_Name => "Switches");

 Switches : GNAT.Strings.String_List_Access :=
  Tree.Root_Project.Attribute_Value (My_Tool_Switches, Index => "Ada");
begin
     for S of Switches loop
        Put_Line (S.all);   --  "-a", then "-b"
     end loop;

     Free (Switches);
end;
```

Некоторые константы для предопределенных атрибутов уже объявлены в GNATCOLL.Projects.

`GNATCOLL.Projects` также предоставляет API для редактирования файлов проекта. Он имеет то же ограничение, что и GPS (что неудивительно): когда вы редактируете проект, изменения могут повлиять на весь файл проекта, поэтому форматирование или комментарии, созданные вручную, могут исчезнуть. В основном вам следует редактировать только те проекты, которые были созданы с помощью одного и того же API, с риском потери пользовательских изменений.

`GNATCOLL.Projects` может анализировать все проекты, даже те, которые он не может редактировать позже, с заметным исключением агрегатных проектов. Поддержка для них есть в планах, но еще не реализована.

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: 3/11/2013

Обсуждение…