Gem #126: Проекты Aggregate Library.

Автор: Pascal Obry—EDF R&D

Краткое содержание: Общепринятая практика создания масштабных приложений на Ada - создание множества модулей. Каждый модуль (или подсистема) имеет свой проектный файл, который необходим не только для поддержки построения модуля, но и помощи в редактировании кода через GPS. Это позволяет разработчикам концентрировать внимание на необходимом им модуле. В целом этот подход упрощает сложность понимания кода приложения, разбивая его на модули.

Давайте начнём…

Общей схемой построения больших приложений Ada является создание нескольких модулей. Каждый модуль (или подсистема) поставляется с файлом проекта, не только для поддержки его построения, но и для облегчения редактирования кода с помощью GPS. Это позволяет разработчикам сосредоточиться только на коде интересующего модуля. В целом это снижает сложность приложения за счет его модуляризации.

Каждый модуль поставляется с библиотечным проектом, используемым для создания библиотеки (общей или статической), содержащей объектный код модуля. Для приложения последний шаг - это просто связывание библиотек для всех модулей. Пока что все хорошо.

Однако, если целью является не создание приложения, а доставка библиотеки (содержащей источники API и объекты), такой подход не очень удобен. Действительно, все библиотеки должны быть скопированы в каталог установки, и файл проекта для каждой библиотеки должен быть распространен для окончательного приложения, чтобы удовлетворить необходимые зависимости. Это громоздко и несколько утомительно, и это еще более утомительно, когда библиотека должна использоваться в приложении, построенном с помощью C или C++, потому что мы накладываем сложность на клиента API, требуя связывания с несколькими библиотеками.

В таких случаях было бы гораздо удобнее распространять единую библиотеку, содержащую весь код.

Возможно, вы сейчас думаете, что это плохая идея-создавать модули и разрабатывать большие приложения, используя плоскую структуру проекта? Вы правы, и именно здесь в игру вступают проекты aggregate library.

Проекты Aggregate library позволяют создавать единую библиотеку (общую или статическую) из набора библиотек или проектов.

Например, предположим, что проект содержит две библиотеки:

```
library project MyLib1 is
   for Source_Dirs use ("src1");
   for Object_Dir use "obj1";
   for Library_Name use "mylib1";
   for Library_Kind use "static";
   for Library_Dir use "lib1";
end MyLib1;

library project MyLib2 is
   for Source_Dirs use ("src2");
   for Object_Dir use "obj2";
   for Library_Name use "mylib2";
   for Library_Kind use "static";
```

```
   for Library_Dir use "lib2";
end MyLib2;
```

Проект `mylib1.gpr` и `mylib2.gpr` можно использовать отдельно для создания `mylib1.a` и `mylib2.a`.

Но с помощью `aggregate library` можно создать одну библиотеку (скажем, `fulllib.a`) объединение обоих проектов:

```
aggregate library project MyLib is
   for Project_Files use ("mylib1.gpr", "mylib2.gpr");
   for Library_Name use "fulllib";
   for Library_Kind use "static";
   for Library_Dir use ".";
end MyLib;
```

Обратите внимание, что описание проекта `aggregate library` очень близко к библиотечному проекту. Единственным новым обязательным атрибутом является `Project_Files`, в котором должны быть перечислены все проекты, которые будут агрегированы.

Обратите внимание также, что атрибут `Object_Dir` не нужен, так как библиотеки `aggregate` сами по себе не имеют объектного кода.

Следующая команда создаст `pck1.o` и `pck2.o` (от `src1/pck1.ads` и `src2/pck2.ads`), используя свои соответствующие проекты, а затем агрегирует полученный объектный код в единую библиотеку с именем "`libfulllib.a`":

```
$ gprbuild -gnat05 -p mylib.gpr
```

Содержимое библиотеки `aggregate` можно проверить с помощью:

```
$ nm libfulllib.a
```

И, конечно, то же самое можно сделать и с разделяемыми библиотеками.

В следующем выпуске мы обсудим инкапсулированные библиотеки, которые весьма полезны для многоязычной разработки.

Связанный со статьёй текст программы

Файл mylib.gpr

```
aggregate library project MyLib is
   for Project_Files use ("mylib1.gpr", "mylib2.gpr");
   for Library_Name use "fulllib";
   for Library_Kind use "static";
   for Library_Dir use ".";
end MyLib;
```

Файл mylib1.gpr

```
library project MyLib1 is
   for Source_Dirs use ("src1");
   for Object_Dir use "obj1";
```

```
   for Library_Name use "mylib1";
   for Library_Kind use "static";
   for Library_Dir use "lib1";
end MyLib1;
```

Файл mylib2.gpr

```
library project MyLib2 is
   for Source_Dirs use ("src2");
   for Object_Dir use "obj2";
   for Library_Name use "mylib2";
   for Library_Kind use "static";
   for Library_Dir use "lib2";
end MyLib2;
```

Файл src1/pck1.ads

```
package Pck1 is
   procedure Call1 is null;
end Pck1;
```

Файл src2/pck2.ads

```
package Pck2 is
   procedure Call2 is null;
end Pck2;
```

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

Об авторе (из общедоступных источников)

Автор Pascal Obry

69, rue Jean Longuet , 92290 Châtenay-Malabry, FRA

Инженер-программист и стремится сделать надежные коды. Pascal Obry также увлечен фотографией.

Навыки

Qualité Logicielle

Langage Ada

Ссылки

http://www.obry.net

http://v2p.fr.eu.org

EDF R&D является отделом исследований и разработок EDF Group.

Last Updated: 10/13/2017
Posted on: 5/28/2012

Обсуждение…