Gem #109: Программные расширения Plug-ins и библиотеки общего пользования Shared Libraries в Ada - Часть 1

Автор: Pascal Obry, EDF R&D

Краткое содержание: В Gem #109 мы обсудим поддержку библиотек общего пользования в GNAT, чтобы проще было понять, как создавать программные расширения в Ada.

Давайте начнём…

Это первая часть из двух частей серии Gem (#109, #110) об Ada плагинов plug-ins в языке программирования Ada. В этом Gem мы обсудим, как GNAT поддерживает разделяемые библиотеки, и взаимодействие между разделяемыми библиотеками и системой времени выполнения Ada. Это необходимо для полного понимания того, как создавать плагины Ada, которые обсуждаются во второй части.

В современных операционных системах редко встречаются полностью автономные приложения, то есть исполняемые файлы, не требующие внешней поддержки со стороны операционной системы. Такие приложения встречаются в основном во встроенном мире. В других доменах приложение построено поверх других служб, доступных из библиотек.

Существуют два основных вида библиотек: статические и динамические. Статическая библиотека - это простой контейнер для объектного кода, который будет включён в исполняемый файл конечного приложения. Это делается компоновщиком в качестве последнего шага создания исполняемого файла. Поэтому такие библиотеки называются статическими (статически связанными, с кодом библиотеки, встроенным в исполняемый файл). Без статических библиотек нам пришлось бы связываться со многими объектными файлами, так что это упрощает связывание. Другим преимуществом статических библиотек, организованных в виде архивов объектных файлов, является возможность выборочной компоновки, так что только необходимые объекты из библиотеки будут связаны в конечный исполняемый файл.

Статическая библиотека может быть построена с помощью GNAT с помощью файлов проекта. Например:

```
library project MyLib is
    for Source_Dirs use ("src");
    for Object_Dir use "obj";
    for Library_Dir use "lib";
    for Library_Name use "mylib";
    for Library_Kind use "static";
end MyLib;
```

Динамические библиотеки – это совсем другая история: они позволяют разделить образ исполняемого файла на несколько частей, некоторые из которых могут быть разделены между различными исполняемыми файлами. Динамические библиотеки предлагают два основных преимущества: 1) совместное использование кода исполняемыми файлами, что уменьшает использование глобальной памяти, и 2) Возможность модульного обслуживания приложения путём замены части приложения без необходимости регенерации или даже прерывания всего приложения. Также важно отметить, что динамические библиотеки требуют прямой поддержки со стороны операционной системы.

Связывание с общей библиотекой выполняется в два этапа. Первый шаг выполняется во время компоновки, когда компоновщик создаёт в исполняемом файле таблицу, содержащую имя общей библиотеки и все символы, которые необходимо импортировать из общей библиотеки. Второй шаг выполняется во время загрузки с помощью динамического компоновщика.

Динамический компоновщик является частью операционной системы и отвечает за завершение компоновки приложения. Этот шаг довольно сложный, но в итоге динамический компоновщик работает следующим образом:

1. Загружает все разделяемые библиотеки, на которые ссылается исполняемый файл.

2. Создаёт разделы данных исполняемых библиотек (только код используется совместно исполняемыми файлами, но не данные).

3. Заполняет соответствующую таблицу разделяемой библиотеки исполняемого файла фактическими адресами связанных символов (переменных и подпрограмм).

4. Вызывает процедуры инициализации (если таковые имеются), определённые в только что загруженных общих библиотеках.

На этом этапе приложение готово к запуску. Конечно, разделяемые библиотеки могут ссылаться на другие разделяемые библиотеки; в этом случае динамический компоновщик рекурсивно выполняет ту же работу. Очень важно понимать, что всякий раз, когда у нас есть общая библиотека Ada, используемая приложением Ada, оба будут ссылаться на одну и ту же систему времени выполнения Ada. Что важно в нашем контексте, так это то, что разработка контролируется исполняемым файлом и вычисляется во время привязки. Это обеспечивает правильную семантику Ada в соответствии со стандартом.

Общую библиотеку можно создать с помощью файла проекта, задав для атрибута `Library_Kind` значение `relocatable`.

Для поддержки подключаемого модуля один и тот же фрагмент кода может загружаться и выгружаться несколько раз, что требует инициализации во время загрузки и привязки к самой библиотеке; это не может зависеть от одноразовой глобальной разработки приложения.

GNAT поддерживает автономные разделяемые библиотеки, и именно эти библиотеки подходят для использования в данном контексте. Автономная библиотека-это библиотека, содержащая необходимый код для рекурсивной разработки модулей Ada, входящих в библиотеку, и всех модулей, от которых она зависит. Поскольку многим библиотекам может потребоваться разработать одни и те же внешние модули, код инициализации должен быть защищён от нескольких инициализаций одних и тех же объектов данных.

Обратите внимание, что эти библиотеки не совсем соответствуют семантике Ada, так как разработка должна происходить только при запуске программы, а не в середине при загрузке плагина. Но такие библиотеки подходят для использования с исполняемыми файлами, написанными на других языках или в качестве подключаемых библиотек.

Объявить библиотеку автономной легко с помощью файлов проекта GNAT: для этого просто нужно добавить дополнительный атрибут в файл проекта библиотеки:

```
for Library_Interface use ("interface_unit1", "interface_unit2", ...);
```

Этот атрибут объявляет программные модули, которые видны из-за пределов библиотеки, и, таким образом, предлагает механизм сокрытия (для всех модулей, не включённых в интерфейсы), дополняя тот механизм сокрытия, который предлагает Ada: любой блок не в интерфейсе не может быть вызван извне. Это означает, в частности, что изменение реализации таких программных модулей можно сделать без перекомпиляции, привязать их или связать их с остальной частью приложения.

Во второй части этой серии мы увидим, как настроить Ada плагин на основе автономных библиотек.

Связанный со статьёй текст программы

Файлов нет

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

Об авторе

Сведений нет.

Last Updated: 11/24/2017
Posted on: 9/5/2011

Обсуждение…