Gem #106: Леди Ada целует питона (Python) - Часть 2

Автор: Emmanuel Briot, AdaCore

Краткое содержание: В данной серии из двух публикаций будет пояснено как применять коллекцию компонентов GNAT для сопряжения кода на языке Ада и кода на языке Python. В первом Gem описывались преимущества такой практики и трудности, с которыми можно столкнуться при прямом сопряжении с библиотеками Python. Во втором Gem будет показано, как начать работать с GNATCOLL для того, чтобы значительно упростить процесс сопряжения.

Давайте начнём…

Первая часть этого Gem описала, почему использование интерфейса Python может обеспечить эффективный способ настройки и расширения вашего приложения. В нем также подчёркивается, как коллекция компонентов GNAT заботится о значительной части работы. Этот Gem теперь войдёт в технические подробности о том, как вы можете использовать GNATCOLL.Scripts на практике. Документация GNATCOLL содержит дополнительную информацию, поэтому, пожалуйста, обратитесь к ней, если вам нужна дополнительная информация (см. GNATColl: GNAT Reusable Components; URL: https://docs.adacore.com/gnatcoll-docs/).

Ваше приложение должно указать, какие языки сценариев он хочет поддерживать. Дескрипторы к этим языкам хранятся в глобальной переменной, называемой Scripts_Repository (которую предоставляет ваше приложение, чтобы её можно было сохранить в вашем собственном типе записи или обернуть защищённым объектом и т. д.).

```
with GNATCOLL.Scripts.Python;
use GNATCOLL.Scripts, GNATCOLL.Scripts.Python;
declare
   Repo : Scripts_Repository := new Scripts_Repository_Record;
begin
   Register_Python_Scripting (Repo, "GPS");
end;
```

Второй параметр в последней строке - это имя модуля Python, который экспортирует ваше приложение. Этот пример извлекается из GPS, где все функции и классы доступны как `GPS.Console`, `GPS.Logger` и т. д. Этот параметр является пространством имён, в котором будут экспортироваться ваши экспортные данные.

Следующим шагом будет регистрация некоторых стандартных классов, которые необходимы самому `GNATCOLL`. Наиболее важным из них является класс `Console`, который обеспечивает ввод / вывод между вашим приложением и Python. Во многих случаях вы не хотите, чтобы Python выводил на `stdout`; например, если вы пишете приложение GUI, для этого вам понадобится интерактивная консоль Python. В результате стандартный ввод / вывод Python будет перенаправлен на экземпляры класса `Console`.

```
Register_Standard_Classes (Repo, "Console");
```

Теперь нам нужно определить, как консоль ведет себя на уровне Ada (например, для обычного `stdin` и `stdout`, или для окна GUI, например). Мы не будем вдаваться в подробности создания пользовательских консолей, но GNATCOLL поставляется с примерами для двух обычных случаев в примерах файлов `examples/textconsole.ads` и `examples/gtkconsole.ads`.

```
Console := GtkConsole.Create (...);  --  See the examples directory
Set_Default_Console
  (Lookup_Scripting_Language (Repo, "python"), Virtual_Console (Console));
```

На этом этапе ваше приложение может отображать окно GUI для взаимодействия с Python, в котором пользователи могут писать обычные команды Python. Но ваше приложение по-прежнему не экспортирует ничего полезного.

Давайте рассмотрим простой пример. Мы хотим экспортировать из Ada функцию, которая выполняет добавление двух целых чисел (конечно, это уже предоставлено Python, но это всего лишь пример). Сначала нам нужно объявить, что мы экспортируем:

```
Register_Command
   (Repo,
    Command => "add",
    Params  => (1 => Param("p1"), 2 => Param("p2"),
                3 => Param("p3", Optional => True)),
    Handler => Handler'Access);
```

Это означает, что из Python мы сможем совершать такие вызовы, как:

```
  n = add(p1=23, p2=45)
  n = add(p2=45, p1=23)   # order of parameters is irrelevant
  n = add(23, 45, 67)     # three parameters
```

Первая версия показывает использование именованных параметров в Python (одна из хороших функций, которые это наследовало от Ada). Эти параметры могут быть определены в любом порядке в Python, и GNATCOLL будет автоматически переупорядочивать их так, чтобы Ваше приложение всегда доступны "`p1`" как первый параметр и "`p2`" как второй параметр.

Как Вы видите в коде Ada, нет никакой ссылки на Python. На самом деле " `add` ", будет доступно на всех зарегистрированных языках (Python здесь, но также и потенциально оболочка GPS, и в будущем другие языки). Если новые языки будут добавлены в GNATCOLL, Вашему коду не будет нужно никакое изменение, чтобы извлечь выгоду из них.

Мы теперь должны обеспечить реализацию:

```
procedure Handler (Data : in out Callback_Data'Class; Command : String) is
   P1, P2 : Integer;
begin
   if Command = "add" then
      P1 := Nth_Arg (Data, 1);
      P2 := Nth_Arg (Data, 2);
      P3 := Nth_Arg (Data, 3, 0);  --  Default value is 0
      Set_Return_Value (Data, P1 + P2 + P3);
   end if;
end Handler;
```

Опять же, этот код не зависит от Python. Мы всегда обращаемся к "`P1`" в качестве первого параметра (через вызов `Nth_Arg`), и GNATCOLL автоматически позаботится о переназначении параметров, если пользователь указал их в другом порядке.

GNATCOLL автоматически вызовет исключение Python, если пользователь вызовет " `add` " с неправильным количеством параметров. Обратите внимание, что также можно экспортировать функции с необязательными параметрами (`P3` в нашем примере). Затем `Nth_Arg` можно использовать для указания значения по умолчанию для параметра.

Также стоит отметить, что `Nth_Arg` вызовет исключение, если соответствующий параметр имеет неправильный Тип. Существует несколько вариантов `Nth_Arg` (для строк, целых чисел, булевых значений и некоторых других типов). В нашем случае, если пользователь вызывает " `add` " со строкой, `Nth_Arg` вызовет исключение Ada, которое может обработать ваше приложение. Если ваше приложение не поймает исключение, оно будет распространено на Python.

Пример в этого Gem имеет ограниченную область применения, но подчёркивает некоторые из основных функций и услуг, которые предлагает GNATCOLL поверх Python. После правильной инициализации GNATCOLL для экспорта новых команд требуется небольшой дополнительный клеевой код. Экспорт классов и функций очень похож на то, что описано в примере. Для получения дополнительной информации см. онлайн-документацию GNATCOLL: Embedding Script Languages;

URL: http://www.adacore.com/wp-content/files/auto_update/gnatcoll-docs/gnatcoll.html#Embedding-script-languages .

На данном этапе наиболее сложной задачей является определение чёткого Python API для вашего приложения, который пользователи могут понять относительно легко, и все же, который может быть расширен в будущем путём экспорта ещё большего количества возможностей из Ada, не нарушая весь дизайн API.

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

Сведений об авторе нет.

Last Updated: 11/24/2017
Posted on: 5/23/2011

Обсуждение…