Gem #102: Клиент SOAP/WSDL

Автор: Pascal Obry, EDF R&D

Краткое содержание: В данном Gem будет продемонстрирована использование веб-услуг в соответствии с описанием, представленным в документе WSDL.

Давайте начнём…

Это вторая часть серии из двух частей Gem для SOAP и WSDL.

В этом Gem мы будем использовать веб-службу, как описано в документе WSDL. Эти службы могут быть реализованы в Java, C # или Ada, поскольку WSDL является универсальным в мире веб-сервисов.

В предыдущем Gem мы создали WSDL из простой спецификации Ada. Давайте используем его для генерации необходимого кода для использования этих веб-сервисов. Мы снова используем инструмент wsdl2aws, но на этот раз генерируем только заглушки:

```
$ wsdl2aws -f -noskel temperatures.wsdl
```

Сгенерирован набор пакетов. На данный момент нас интересуют два, а именно:

Пакет temperature_service-types.ads, содержащий типы, используемые веб-службами.

Пакет temperature_service-client.ads, содержащий спецификацию клиента Web-сервисов.

Для каждой процедуры веб-службы генерируются две спецификации:

```
function To_Fahrenheit
  (C        : Celsius_Type;
   Endpoint : String := Temperatures_Service.URL;
   Timeouts : AWS.Client.Timeouts_Values := Temperatures_Service.Timeouts)
   return To_Fahrenheit_Result;

function To_Fahrenheit
  (Connection : AWS.Client.HTTP_Connection;
   C          : Celsius_Type)
   return To_Fahrenheit_Result;

--  Raises SOAP.SOAP_Error if the operation fails
```

Первый подключает и закрывает соединение для каждого вызова, тогда как второй использует постоянное соединение. Это просто. Теперь давайте построим небольшую программу, которая преобразует значения градусов по Цельсию в градусы по Фаренгейт:

```
with Ada.Text_IO;
with Temperatures_Service.Client;
with Temperatures_Service.Types;

procedure SOAP_Client is
   use Ada;
   use Temperatures_Service;
   C : constant Types.Celsius_Type := 20.0;
   F : constant Types.Fahrenheit_Type := Client.To_Fahrenheit (C);
   package C_IO is new Text_IO.Float_IO (Types.Celsius_Type);
   package F_IO is new Text_IO.Float_IO (Types.Fahrenheit_Type);
```

```
begin
   Text_IO.Put ("Celsius    "); C_IO.Put (C, Aft => 1, Exp => 0);
   Text_IO.New_Line;
   Text_IO.Put ("Fahrenheit "); F_IO.Put (F, Aft => 1, Exp => 0);
   Text_IO.New_Line;
end SOAP_Client;
```

Мы можем использовать следующий простой файл проекта для создания этой программы:

```
with "aws";
project SOAP_Client is
   for Source_Dirs use (".");
   for Main use ("soap_client.adb");
end SOAP_Client;
$ gnatmake -gnat05 -Psoap_client
```

Теперь давайте протестируем его, сначала запустив сервер, который мы создали на прошлом неделе в Gem #101:

```
$ ./soap_server
```

Затем запустите `soap_client`:

```
$ ./soap_client
Celsius    20.0
Fahrenheit 68.0
```

Вот и все. Как мы показали, легко использовать веб-службу в Ada при предоставлении WSDL. По-прежнему можно использовать веб-службу без WSDL, но в этом случае необходимо будет вручную её обработать.

Связанный со статьёй текст программы

soap_client.zip

```
soap_client.gpr

with "aws";
project SOAP_Client is
   for Source_Dirs use (".");
   for Main use ("soap_client.adb");
end SOAP_Client;

soap_client.adb

.1.
 2. with Ada.Text_IO;
 3.
 4. with Temperatures_Service.Client;
 5. with Temperatures_Service.Types;
 6.
 7. procedure SOAP_Client is
 8.    use Ada;
 9.    use Temperatures_Service;
10.    C : constant Types.Celsius_Type := 20.0;
11.    F : constant Types.Fahrenheit_Type := Client.To_Fahrenheit (C);
12.    package C_IO is new Text_IO.Float_IO (Types.Celsius_Type);
13.    package F_IO is new Text_IO.Float_IO (Types.Fahrenheit_Type);
```

```
14. begin
15.    Text_IO.Put ("Celsius    "); C_IO.Put (C, Aft => 1, Exp => 0);
16.    Text_IO.New_Line;
17.    Text_IO.Put ("Fahrenheit "); F_IO.Put (F, Aft => 1, Exp => 0);
18.    Text_IO.New_Line;
19. end SOAP_Client;
```

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

Об авторе

Сведений нет.

Last Updated: 10/13/2017
Posted on: 3/28/2011

Обсуждение…