Gem #29: Введение в Веб-Серверы на языке Ada (AWS)

Автор: Pascal Obry, EDF R&D

Краткое содержание: Gem Ада #29 - Введение в Веб-Серверы Ада (AWS).

Давайте начнем…

Этот Gem представляет собой базовое введение в веб-сервер Ada (AWS). Основные компоненты AWS включают веб-сервер, который поддерживает протокол HTTP. Этот веб-сервер может быть встроен в любое приложение. Обычными являются:

1. - Разработать полное веб-приложение.
2. - Добавление графического интерфейса в основном пакетно-ориентированное приложение, такое как, например длительное моделирование, с которым мы хотим время от времени взаимодействовать.
3. - Разработать распределённое приложение, обменивающееся сообщениями с использованием протоколов HTTP или SOAP.

Модуль AWS HTTP обрабатывает декодирование HTTP-запроса и кодировку ответа пользователя. Он использует механизм обратного вызова для взаимодействия с приложением пользователя. Возьмем знаменитый пример Hello World и посмотрим, как он может быть переведен в рамки AWS.

Сначала обратный вызов hello_world пользователя, который является функцией с единственным параметром типа Status.Data и возвращает объект Response.Data:

```
with AWS.MIME;
with AWS.Response;
with AWS.Status;

package body CB is
   use AWS;
   function Hello_World (Request : in Status.Data) return Response.Data is
   begin
      return Response.Build (MIME.Text_HTML, "<p>Hello World!</p>");
   end Hello_World;
end CB;
```

Модуль AWS.Response содержит множество конструкторов. Response.Build - один. Другой - Response.File, для возврата файла. Предусмотрены различные конструкторы высокого уровня, например, для потоковой передачи данных.

Теперь вот основная подпрограмма с сервером:

```
with AWS.Server;
with CB;
procedure Hello_World is
   use AWS;
   HTTP : Server.HTTP;
begin
   Server.Start (HTTP, "Hello_World", Callback => CB.Hello_World'Access);
   delay 60.0;
   Server.Shutdown (HTTP);
end Hello_World;
```

Поясним. Подпрограмма запуска регистрирует процедуру CB.Hello_World как обратный вызов пользователя и запускает сервер, используя порт по умолчанию, который является 8080. После запуска исполняемого файла программы hello_world, вводя URL http://localhost:8080/ в Ваш веб-браузер Вы получите сообщение «Hello World», и это произойдёт для любого URI. Так ввод http://localhost:8080/whatever также возвратит «Hello World». После 60 секунд закроется сервер, и программа выйдет из системы.

Все параметры HTTP доступны из параметра Request для обратного вызова Hello_World. Поэтому давайте добавим локальную константу, чтобы получить фактический URI:

```
URI : constant String := Status.URI (Request);
```

Затем мы меняем вызов Response.Build:

```
return Response.Build
   (MIME.Text_HTML, "<p>Hello World! URI=" & URI & "</p>");
```

После перезапуска сервера, введя http://localhost:8080/home в веб-браузер, отобразится «Hello World! URI=/home».

Конструкция if / elsif может использоваться для проверки каждого URI, который должен обрабатываться сервером:

```
if URI = "…" then
   …
elsif URI = "…" then
   …
else
   return Response.Build
      (MIME.Text_HTML,
       "<p>Not found : URI=" & URI & "</p>",
       Status_Code => Messages.S404);
end if;
```

Использование обратных вызовов прост, но когда приложение становится больше, проще использовать диспетчеров. Итак, давайте изменим программу Hello World для использования диспетчеров:

```
with AWS.Response;
with AWS.Status;
with AWS.Dispatchers;
package CB is
  use AWS;

  type Hello_World is new Dispatchers.Handler with null record;

  overriding function Dispatch
    (Handler : in Hello_World;
     Request : in Status.Data) return Response.Data;
private
  overriding function Clone (Element : in Hello_World) return Hello_World;
end CB;
```

Тело Dispatch идентично обратному вызову Hello_World выше. Тип Dispatchers.Handler имеет клонируемый интерфейс, а Clone в этом случае тривиален:

```
   overriding function Clone (Element : in Hello_World) return Hello_World is
   begin
      return Element;
   end Clone;
```

В этом случае основная подпрограмма немного больше:

```
   with AWS.Config;
   with AWS.Server;
   with AWS.Services.Dispatchers.URI;
   with CB;

   procedure Hello_World is
      use AWS;
      HTTP : Server.HTTP;
      Conf : Config.Object;
      HW   : CB.Hello_World;
      Root : Services.Dispatchers.URI.Handler;
   begin
      Services.Dispatchers.URI.Register (Root, "/hello", HW);
      Server.Start (HTTP, Root, Conf);
      delay 60.0;
      Server.Shutdown (HTTP);
  end Hello_World;
```

Основное различие заключается в том, что мы зарегистрировали диспетчер HW, который будет использоваться только для URI «/hello» (http://localhost:8080/hello). Для любого другого URL-адреса диспетчерский модуль отправляет сообщение об ошибке 404. При необходимости также можно использовать регулярное выражение или префикс. Объект конфигурации может быть настроен на изменение любых параметров сервера, таких как порт, количество одновременных подключений, тайм-ауты и т. д.

Существует множество типов диспетчеров не только для поддержки URI (как в этом примере), но и для обработки виртуальных хостов, методов запросов (POST / GET) и служб для связывания диспетчеров, а также диспетчеризации на основе таймеров (см. child units of AWS.Services.Dispatchers). Пример простого API для преобразования обратного вызова диспетчеру см. В AWS.Dispatchers.Callback.

Различные другие услуги предлагаются непосредственно AWS, включая HTTP/SOAP, WSDL, Ajax, SMTP и механизм шаблонов. Но охват всех этих вопросов выходит за рамки этого Gem-29.

Связанный исходный код

Attached Files

Файл `prj1.gpr`

```
with "aws";

project Prj1 is
   for Source_Dirs use ("src1");
   for Object_Dir use "obj1";
   for Main use ("hello_world.adb");
   package Builder is
      for Default_Switches ("Ada") use ("-gnat05");
   end Builder;
end Prj1;
```

Файл hello_world.adb

```
with AWS.Server;
with CB;

procedure Hello_World is
   use AWS;
   HTTP : Server.HTTP;
begin
   Server.Start (HTTP, "Hello_World", Callback => CB.Hello_World'Access);
   delay 60.0;
   Server.Shutdown (HTTP);
end Hello_World;
```

Файл cb.ads

```
with AWS.Response;
with AWS.Status;

package CB is

   use AWS;

   function Hello_World (Request : in Status.Data) return Response.Data;

end CB;
```

Файл cb.adb

```
with AWS.MIME;
with AWS.Parameters;

package body CB is

   function Hello_World (Request : in Status.Data) return Response.Data is
      URI : constant String := Status.URI (Request);
   begin
   return Response.Build
      (MIME.Text_HTML, "<p>Hello World! URI=" & URI & "</p>");
   end Hello_World;

end CB;
```

Файл prj2.gpr

```
with "aws";

project Prj2 is
   for Source_Dirs use ("src2");
   for Object_Dir use "obj2";
   for Main use ("hello_world.adb");
   package Builder is
      for Default_Switches ("Ada") use ("-gnat05");
   end Builder;
end Prj2;
```

Файл hello_world.adb

```
with AWS.Config;
with AWS.Server;
with AWS.Services.Dispatchers.URI;
with CB;
```

```
procedure Hello_World is
   use AWS;
   HTTP : Server.HTTP;
   Conf : Config.Object;
   HW   : CB.Hello_World;
   Root : Services.Dispatchers.URI.Handler;
begin
   Services.Dispatchers.URI.Register (Root, "/hello", HW);
   Server.Start (HTTP, Root, Conf);
   delay 60.0;
   Server.Shutdown (HTTP);
end Hello_World;
```

Файл `cb.ads`

```
with AWS.Response;
with AWS.Status;
with AWS.Dispatchers;

package CB is

   use AWS;

   type Hello_World is new Dispatchers.Handler with null record;

   overriding function Dispatch
     (Handler : in Hello_World;
      Request : in Status.Data) return Response.Data;

private

   overriding function Clone (Element : in Hello_World) return Hello_World;

end CB;
```

Файл `cb.adb`

```
with AWS.MIME;
with AWS.Parameters;

package body CB is

   overriding function Clone (Element : in Hello_World) return Hello_World is
   begin
      return Element;
   end Clone;

   overriding function Dispatch
     (Handler : in Hello_World;
      Request : in Status.Data) return Response.Data
   is
      URI : constant String := Status.URI (Request);
   begin
      return Response.Build
        (MIME.Text_HTML, "<p>Hello World! URI=" & URI & "</p>");
   end Dispatch;

end CB;
```

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