Gem #21: Анализ XML текста

Автор: Emmanuel Briot, Senior Software Engineer, AdaCore

Краткое содержание: Gem Ада #21 - Консорциум World Wide Web (W3C) разрабатывает различные спецификации на формат файлов XML. В частности, определены различные API для загрузки, обработки и записи файлов XML. Несмотря на то, что для языка Ада API не определены, XML/Ада соответствует им настолько, насколько возможно. В данном gem описывается, как использовать XML/Ада для анализа XML файлов..

Давайте начнем…

Существует два основных API для анализа XML-файл. Один из них - DOM (документ объект модель, DOM) читает этот файл и создает дерево в памяти, представляющим весь документ. Как правило, из-за объема операций, санкционированных спецификациями, это дерево в несколько раз больше, чем сам документ, и, таким образом, в зависимости от объема памяти на компьютере, это может ограничить размер документов, которые приложение может прочитать. С другой стороны, она обеспечивает большую гибкость в обращении с этими деревьями.

Другой метод (SAX) основан на обратных вызовах, которые вызываются при просмотре различных конструкций при чтении XML-файла. Это почти не требует памяти, но делает обработку XML-файла дополнительной работой для вашего приложения. Однако он очень хорошо подходит, когда вы хотите хранить XML-данные в структуре данных приложения. Фактически, сам XML / Ada использует SAX для построения дерева DOM.

В обоих случаях, XML/Ada нужен объект (“input_source”) читать фактические данные XML. Эти данные можно найти либо на диске, в памяти, чтение из сокета или любого другого источника вы можете себе представить. XML/Ada тщательно сконструирован таким образом, что он не требует всего документа в памяти, и может просто читать по одному символу за раз, что делает его адаптируемым к любому возможному входу. Этот GEM не охватывает, как написать свои собственные входные потоки. Это в целом довольно легко, единственная сложность заключается в правильной конвертации байтов, которые вы читаете в символы Юникода.

Ниже приведен небольшой пример использования API DOM для создания дерева в памяти. В этом примере предполагается, что наиболее часто встречается XML-файл на диске, поэтому в качестве входных данных используется File_Input. Второй объект, который нам нужен сам парсер XML. Когда мы хотим создать дерево DOM, мы должны использовать Tree_Reader, или тип, производный от него. Как мы увидим позже, это по сути синтаксический анализатор SAX (то есть событие на основе XML-парсер), чьи вызовы осуществляются для создания дерева DOM. Можно, конечно, переопределить его примитивные операции, если вы хотите сделать дополнительные вещи (например, подробный вывод, перенаправление сообщений об ошибках, предварительная обработка XML-узлов,...).

```
with Input_Sources.File;  use Input_Sources.File;
with DOM.Readers;         use DOM.Readers;
with DOM.Core;            use DOM.Core;

procedure Read_XML_File (Filename : String) is
  Input  : File_Input;
```

```
  Reader : Tree_Reader;
  Doc    : Document;
begin
  Open (Filename, Input);
  Parse (Reader, Input);
  Close (Input);

  Doc := Get_Tree (Reader);
  ...
  Free (Reader);
end Read_XML_File;
```

Первые три строки читают файл в памяти. Четвертая строка получает дескриптор самого дерева, который затем можно манипулировать с помощью различных подпрограмм, найденных в пакетах DOM.Core.* (И которые предусмотрены спецификациями W3C). Когда мы закончим, мы просто освободим память.

Есть различные параметры, которые можно задать на ридере перед анализом XML-потока, например, следует ли поддерживать пространства имен XML, нужно ли проверять входные данные и так далее.

Как мы уже упоминали ранее, существует второй API нижнего уровня под названием SAX, основанный на событиях. Он определяет один теговый тип, Reader, который имеет несколько примитивных операций, которые действуют как обратные вызовы. Вы можете переопределить те, которые вы хотите. Как правило, результатом их вызова является создание в памяти представления входных данных XML (что и делает интерфейс DOM, на самом деле).

Следующий короткий пример только обнаруживает запуск элементов в XML-файле и распечатывает их имя на стандартном выводе. Это имеет мало интереса к реальным приложениям, но является хорошей платформой, на которой можно базировать Ваши собственные синтаксические анализаторы SAX.

```
with Sax.Attributes;
with Sax.Readers;     use Sax.Readers;
with Unicode.CES;     use Unicode.CES;

package Debug_Parsers is
   type Debug_Reader is new Reader with null record;
   overriding procedure Start_Element
     (Handler       : in out Debug_Reader;
      Namespace_URI : Unicode.CES.Byte_Sequence := "";
      Local_Name    : Unicode.CES.Byte_Sequence := "";
      Qname         : Unicode.CES.Byte_Sequence := "";
      Atts          : Sax.Attributes.Attributes'Class);
end Debug_Parsers;
```

Вот реализация обратного вызова Start_Element. Мы предполагаем, в этом простом примере, что консоль, на которой мы печатаем вывод, может принимать символы Юникода (на самом деле, все Put_Line делает, чтобы напечатать серию байтов, которые интерпретируются консолью, чтобы сделать правильный рендеринг глифов Юникода).

```
with Ada.Text_IO;   use Ada.Text_IO;

package body Debug_Parsers is
   procedure Start_Element
     (Handler       : in out Debug_Reader;
      Namespace_URI : Unicode.CES.Byte_Sequence := "";
      Local_Name    : Unicode.CES.Byte_Sequence := "";
      Qname         : Unicode.CES.Byte_Sequence := "";
```

```
      Atts          : Sax.Attributes.Attributes'Class)
   is
   begin
      Put_Line ("Found start of " & Qname);
   end Start_Element;
end Debug_Parsers;
```

И, наконец, вот краткий пример программы, использующей этот синтаксический анализатор. Обратите внимание на то, как он точно имитирует то, что мы сделали для DOM (что не удивительно, поскольку, опять же, парсер DOM сам по себе является особой реализацией анализатора SAX).

```
with Input_Sources.File;  use Input_Sources.File;
with Debug_Parsers;       use Debug_Parsers;

procedure Test_Sax is
  Input  : File_Input;
  Reader : Debug_Reader;
begin
  Open (Filename, Input);
  Parse (Reader, Input);
  Close (Input);
end Test_Sax;
```

Связанный исходный код

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

Обсуждение…

Даниал

11 мая 2012

Привет Эммануил, еще один отличный пост. Лично я предпочитаю метод SAX, но я также нашел его легче разбирать с помощью анализатора, как liquid studio (http://www.liquid-technologies.com), каков ваш опыт работы с инструментами синтаксического анализа?