XML сериализация объекта Delphi

Автор: Чудин Андрей

В статье рассмотрены возможности прямой загрузки/сохранения XML документов в объекты Delphi/C++Builder и генерации соответствующих DTD. Предлагается оптимизированный компонент для реализации этих возможностей.

Язык XML предоставляет нам чрезвычайно удобный и почти универсальный подход к хранению и передаче информации. Существует множество парсеров для разбора XML документов по модели DOM. На платформе Microsoft Windows - это, в первую очередь, парсеры MSXML от Microsoft.

Парсеры взаимодействуют с вызывающими приложениями посредством интерфейса SAX (Simple API for XML) и/или DOM (Document Object Model). Во всех анализаторах, за исключением продукта фирмы Microsoft, используется SAX, и почти во всех их возможно применение DOM.

Реализация парсера MSXML не плоха, поддерживает проверку семантической корректности документа и с его помощью достаточно удобно загружать небольшие XML документы. Однако для работы с каждым типом документов, реализованном на XML разработчику приходится создавать некий оберточный код для загрузки данных из объекта Microsoft.XMLDOM во внутренние структуры программы или для удобного перемещения по DOM. При изменении формата документа, что часто возможно в части расширения его спецификации, изменения созданного кода могут быть достаточно трудоемкими и требующими тщательной отладки.

Возникает вопрос возможности упростить работу с XML документами, интегрировать их обработку в разрабатываемые программы. Для модели DOM наилучшим является непосредственная загрузка XML документа в объект Delphi/C ++Builder. И эта возможность есть. Используя RTTI можно загружать данные непосредственно из тегов XML документа в атрибуты заданного объекта. Соответственно, становится возможным и XML-сериализация published интерфейсов объектов любых классов Delphi.

Рассматриваемый подход дает возможность наиболее удобно интегрировать обработку XML в среду разработки Delphi и C++Builder. Возможность доступа к свойствам объектов определяется через механизмы RTTI. Его возможности в Delphi очень велики, т.к. среда разработки сама хранит ресурсы объектов в текстовом формате.

Очевидно, что за предлагаемыми преимуществами скрываются и ряд ограничений. В первую очередь, это касается атрибутов тегов. У нас нет простых механизмов отличить атрибут от тега при сохранении свойства объекта. Поэтому в предлагаемой реализации мы будем обрабатывать XML документы, не содержащие атрибутов. Это ограничение может стать критическим только если мы хотим поддержать уже существующий тип XML документа. Если же мы разрабатываем формат сами, то вполне можем отказаться от атрибутов. Зато наш парсер будет работать не просто быстро, а очень быстро. ;)

Алгоритм XML-сериализации реализуется в виде рекурсивного обхода published интерфейса объекта. Для начала определим ряд простых функций для формирования XML кода. Они позволят нам добавлять открывающие, закрывающие теги и значения в выходной поток.

```
{ пишет строку в выходящий поток. Исп-ся при сериализ
ации }

procedure WriteOutStream(Value: string);
begin
  OutStream.Write(Pchar(Value)[0], Length(Value));
end;

{ Добавляет открывающий тег с заданным именем }

procedure addOpenTag(const Value: string);
begin
  WriteOutStream(CR + DupStr(TAB, Level) + '<' + Valu
e + '>');
  inc(Level);
end;

{ Добавляет закрывающий тег с заданным именем }

procedure addCloseTag(const Value: string; addBreak:
boolean = false);
begin
  dec(Level);
  if addBreak then
    WriteOutStream(CR + DupStr(TAB, Level));
  WriteOutStream('</' + Value + '>');
end;

{ Добавляет значение в результирующую строку }


procedure addValue(const Value: string);
begin
  WriteOutStream(Value);
end;
```

```
end;
```

Следующее, что предстоит реализовать - это перебор всех свойств объекта и формирование тегов. Сведения о свойствах получаются через интерфейс компонента. Это информация о типе. Для каждого свойства, за исключением классовых получается их имя и текстовое значение, после чего формируется XML-тег. Значение загружается через ф-ию TypInfo.GetPropValue();

```
procedure TglXMLSerializer.SerializeInternal(Componen
t: TObject;
                                              Level: i
nteger = 1);
var
  PropInfo: PPropInfo;
  TypeInf, PropTypeInf: PTypeInfo;
  TypeData: PTypeData;
  i, j: integer;
  AName, PropName, sPropValue: string;
  PropList: PPropList;
  NumProps: word;
  PropObject: TObject;
begin
  { Playing with RTTI }
  TypeInf := Component.ClassInfo;
  AName := TypeInf^.Name;
  TypeData := GetTypeData(TypeInf);
  NumProps := TypeData^.PropCount;

  GetMem(PropList, NumProps * sizeof(pointer));
  try
    { Получаем список строк }
    GetPropInfos(TypeInf, PropList);

    for i := 0 to NumProps - 1 do
    begin
      PropName := PropList^[i]^.Name;

      PropTypeInf := PropList^[i]^.PropType^;
      PropInfo := PropList^[i];

      case PropTypeInf^.Kind of
        tkInteger, tkChar, tkEnumeration, tkFloat, tk
String, tkSet,
          tkWChar, tkLString, tkWString, tkVariant:
          begin
            { Получение значения свойства }
            sPropValue := GetPropValue(Component, Pro
pName, true);

            { Перевод в XML }
            addOpenTag(PropName);
            addValue(sPropValue); { Добавляем значени
е свойства в результат }
            addCloseTag(PropName);
          end;
```

Для классовых типов придется использовать рекурсию для

загрузки всех свойств соответствующего объекта.

Более того, для ряда классов необходимо использовать особый подход. Сюда относятся, к примеру, строковые списки и коллекции. Ими и ограничимся.

Для текстового списка TStrings будем сохранять в XML его свойство CommaText, а в случае коллекции после обработки всех ее свойств сохраним в XML каждый элемент TCollectionItem отдельно. При этом в качестве контейнерного тега будем использовать имя класса TCollection(PropObject).Items[j].ClassName.

```
tkClass: { Для классовых типов рекурсивная обработка
}
begin
  addOpenTag(PropName);

  PropObject := GetObjectProp(Component, PropInfo);
  if Assigned(PropObject) then
  begin
    { Для дочерних свойств-классов - рекурсивный вызо
в }
    if (PropObject is TPersistent) then
      Result := Result + SerializeInternal(PropObject
, Level);

    { Индивидуальный подход к некоторым классам }
    if (PropObject is TStrings) then { Текстовые спис
ки }
    begin
      WriteOutStream(TStrings(PropObject).CommaText);
    end
    else if (PropObject is TCollection) then { Коллек
ции }
    begin
      Result := Result + SerializeInternal(PropObject
, Level);
      for j := 0 to (PropObject as TCollection).Count
 - 1 do
      begin
        addOpenTag(TCollection(PropObject).Items[j].C
lassName);
        SerializeInternal(TCollection(PropObject).Ite
ms[j], Level);
        addCloseTag(TCollection(PropObject).Items[j].
ClassName, true);
      end
    end;
    { Здесь можно добавить обработку остальных классо
в: TTreeNodes, TListItems }

  end;
  addCloseTag(PropName, true);
end;
```

Описанные функции позволят нам получить XML код для объекта включая все его свойства. Остается только 'обернуть' полученный XML в тег верхнего уровня - имя класса объекта. Если мы

XML в тэг верхнего уровня - имя класса объекта. Если мы поместим вышеприведенный код в функцию SerializeInternal(), то результирующая функция Serialize() будет выглядеть так:

```
procedure Serialize(Component: TObject; Stream: TStre
am);
...
  WriteOutStream(PChar(CR + '<' + Component.ClassName
 + '>'));
SerializeInternal(Component);
WriteOutStream(PChar(CR + '</' + Component.ClassName
+ '>'));
```

К вышеприведенному можно добавить еще ф-ии для форматирования генерируемого XML кода. Также можно добавить возможность пропуска пустых значений и свойств со значениями по умолчанию. Все эти расширения мы реализуем при создании готового компонента.

Следует заметить, что при желании можно переписать этот код для генерации также и атрибутов элементов. Для отличия элементов от их атрибутов в интерфейсе сохраняемого объекта можно принять следующее соглашение: элементами являются только классовые типы, все же прочие свойства кодируются как атрибуты соответствующих классов. Соответственно можно модифицировать и парсер. При этом появляется возможность использования XML схем вместо DTD. Тут, однако, возникает проблема описания модели содержания для текста #PCDATA. Для разрешения проблемы придется выделить отдельный класс для хранения подобных данных.

Оставить комментарий

Комментарий:

можно использовать BB-коды

Максимальная длина комментария - 4000 символов.

CodeNet ВКонтакте Facebook Twitter Google Яндекс

Добавить

Комментарии

1. Аноним

18 ноября 2004, 07:05:04 +1 / -0

Всё прикольно, но как быть, если мне нужно сохранить свой компонент, например TPanel, а внутри мои контролы?