УДК 004.4′ 2 4

М.Н. КРАВЦОВ

Харьковский национальный университет имени В.Н. Каразина, Украина

КОДОГЕНЕРИРУЮЩИЙ ИНСТРУМЕНТАРИЙ РАЗРАБОТЧИКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Рассмотрен подход к повышению эффективности процесса разработки программного обеспечения путем создания и использования кодогенерирующих инструментальных средств. Рассмотрены технические приемы, позволяющие минимизировать участие разработчика в процессе создания ПО с использованием кодогенерирующих инструментов.

разработка ПО, кодогенерация, интроспекция, шаблон, XML, XSLT, СУБД, метаданные

Введение

В процессе разработки программного обеспечения программист нередко сталкивается со своеобразной рутиной – написанием тривиального и шаблонного кода, который занимает много времени, большой по объему и, соответственно, может быть потенциальным контейнером для множества ошибок вызванным человеческим фактором. Применение инструментальных средств для генерации такого кода помогает избежать ошибок и экономит время разработчика, но имеет и противовес – в случае, если подходящий инструмент не существует, его приходится разрабатывать, затрачивая время и ресурсы [1]. Также генерация кода может использоваться для избежания повторного написания кода, являясь, в данном аспекте, альтернативой компонентному программированию. Строго говоря, автоматически генерировать можно не только исходный код, но и документацию, отчеты, конфигурационные файлы, тестовые данные и т.п. Для решения такого рода задач процесс генерации кода целесообразно разбить на два этапа: сбор метаданных и собственно генерацию в соответствии с определенным набором правил [2]. На этапе сбора метаданных возможно применение различных приемов автоматизации, таких как интроспекция схемы баз данных с использованием системных процедур СУБД или анализ структуры управляемого кода с использованием специальных функций платформы, для которой разрабатывается приложение.

Формулировка проблемы. Процесс ручного кодирования занимает много времени разработчика, приводит к утомлению, особенно если производимый код носит тривиальный и шаблонный характер. Произведенный разработчиком код, как правило, нуждается в отладке, которая тоже требует времени и усилий. Рассмотрим пример прикладной задачи: требуется сгенерировать код для CRUD – процедур в базе данных, СУБД – Microsoft SQL Server, целевой язык – Transact-SQL, платформа для разработки приложений – Microsoft .NET, язык – C#.

Решение проблемы

Собственно генерацию целесообразно проводить с использованием технологий, позволяющих удобно производить необходимые преобразования. Это может быть как популярная связка технологий XML/XSLT [3, 4], так и уже разработанные библиотеки для шаблонных преобразований, например StringTemplate [4]. Ну и, разумеется, всегда есть достаточно прямой вариант – когда преобразующий модуль пишется полностью вручную.

Сбор и представление метаданных. Ключевым элементом нашего генератора будет класс TableDescriptor, который будет содержать метаданные, необходимые для генерации кода. Для генерации CRUD-процедур нам необходимо знать следующую информацию о таблице: ее название, перечень полей

и типов их данных, перечень полей, входящих в первичный ключ таблицы. Соотвественно объявлен этот класс может быть следующим образом:

```
public class TableDescriptor
{
      private string _Name;
      public string Name
      {
            get { return _Name;}
      }
public string[] PrimaryKeyColumns;
      public DataColumn[] Columns{…}
            private ArrayList _cols;
        public TableDescriptor(){…}
}
```

Собрать метаданные мы можем автоматически, используя системные процедуры СУБД SQL Server. Для примера соберем метаданные о таблице Products, находящиеся в базе данных Northwind, запрос в этом случае может выглядеть следующим образом:

```
use northwind
select
      sc.name,
  st.name type,
  sc.length
from
      sysobjects,syscolumns
      sc,systypes st
  where sysobjects.name='Products'
  and sc.id=sysobjects.id
  and sc.xtype = st.xtype
order by sc.colorder
```

Результат выполнения этого запроса, если, конечно, база не подвергалась изменениям после установки, представлен в табл. 1.

Таблица 1

Результат выполнения запроса

	name	type	length
1	ProductID	int	4
2	ProductName	nvarchar	80
3	ProductName	sysname	80
4	SupplierID	int	4
5	CategoryID	int	4
6	QuantityPerUnit	nvarchar	40
7	QuantityPerUnit	sysname	40
8	UnitPrice	money	8
9	UnitsInStock	smallint	2
10	UnitsOnOrder	smallint	2
11	ReorderLevel	smallint	2
12	Discontinued	bit	1

После получения данных в таком формате, перегнать их в объект типа TableDescriptor не составит никакого труда.

Генерация кода. На этом этапе мы преобразовываем наши метаданные в исходный код, рассмотрим три возможных способа реализации такого преобразования.

Прямая генерация кода. При прямой генерации кода мы вручную формируем нужный нам текст пользуясь обычными функциями платформы для разработки приложений, в данном случае – .NET [3]. Исходный код для генерации тела INSERT процедуры может выглядеть так:

```
public class StraightGenerator
{
      public static string Gener-
ateInsert (TableDescriptor t)
            {
            StringBuilder res = new
StringBuilder();
      StringBuilder  paramlist  =  new
StringBuilder();
StringBuilder paramtypelist = new
StringBuilder();
foreach(DataColumn d in t.Columns){
                  param-
list.Append("\t@"+d.ColumnName
+",\n");}
      param-
list.Remove(paramlist.Length -2,1);
      res.AppendFormat("INSERT
INTO\n\t{0} \nVALUES
\n(\n{1})",t.Name,paramlist.ToString(
));
return res.ToString();
            }
      }
```

Очевидно, подход не очень хорош, код, формирующий исходный текст, выглядит довольно неаккуратно и сложен для восприятия. Однако у этого варианта есть положительные стороны:

– разработчик не тратит время на освоение XML/XSLT или StringTemplate, или какого-то иного дополнительного средства для шаблонных преобразований;

– если генерируемый код очень прост, то быстрее написать его вручную , чем подключать дополнительные средства, сохраняя при этом относительную прозрачную структуру генерирующей процедуры.

Генерация кода с помощью XML/XSLT. Для реализации этого подхода нужен исходный XML для преобразования с помощью XSLT. Мы можем его либо сгенерировать из объектного представления, либо получить прямо из SQL Server'а, воспользовавшись директивой FOR XML. В любом случае исходный XML будет выглядеть так:

```
<table name="Products">
      <column name="ProductID"
type="int" length="4"></column>
      <column name="ProductName"
type="nvarchar" length="80"></column>
      <column name="SupplierID"
type="int" length="4"></column>
      <column name="CategoryID"
type="int" length="4"></column>
      <column name="QuantityPerUnit"
type="nvarchar" length="40"></column>
      <column name="UnitPrice"
type="money" length="8"></column>
      <column name="UnitsInStock"
type="smallint" length="2"></column>
      <column name="UnitsOnOrder"
type="smallint" length="2"></column>
      <column name="ReorderLevel"
type="smallint" length="2"></column>
      <column name="Discontinued"
type="bit"
length="1"></column></table>
```

Фрагмент шаблона для генерации теля процедуры Insert будет выглядеть соответственно так:

```
      <template match="/table">
      INSERT INTO
                  <value-of se-
lect="@name"/>
      VALUES
      (
            <for-each se-
lect="column">@<value-of se-
lect="@name"/><if test="position() !=
last()">,</if>&#10;&#09;&#09;</for-
each>
      )
      </template>
```

Преимущество использования XML/XSLT для кодогенерации состоит, в основном, в прозрачности XML, правильность промежуточного представления на XML достаточно просто проверить визуально, чего, к сожалению, нельзя сказать об XSLT.

Еще один недостаток – затраты на освоение XML и XSLT.

*Генерация кода с помощью StringTemplate***.** Шаблон для StringTemplate может иметь следующий вид:

```
INSERT INTO
      $tabledescriptor.name$
VALUES
(
      $first(tabledescriptor.columns)
:{\t\t@$it.ColumnName$,\n}$
      $rest(tabledescriptor.columns):
{\t\t@$it.ColumnName$,\n}$

      $last(tabledescriptor.columns):
{\t\t@$it.ColumnName$\n}$
)
```

Специализированные библиотеки вроде StringTemplate представляют достаточно широкие возможности для применения шаблонов, в случае если генерируемый код сложен по структуре, такие библиотеки могут быть лучшим выбором для создания модульного и прозрачного кодогенератора.

Однако, как и в случае с XML/XSLT, требуют времени на изучение и освоение.

К тому же, open source проекты, такие как StringTemplate, часто страдают от недостатка документации и большого количества ошибок в самой библиотеке [5].

Результат всех трех вариантов реализации идентичен:

```
INSERT INTO
        Products
VALUES
(
    @ProductID,
    @ProductName,
    @SupplierID,
    @CategoryID,
    @QuantityPerUnit,
    @UnitPrice,
    @UnitsInStock,
    @UnitsOnOrder,
    @ReorderLevel,
    @Discontinued
)
```

Разумеется, писать специализированное инструментальное средство, если требуется лишь пара процедур, не стоит, но когда объем кода достаточно велик, то кодогенерирующие инструменты себя окупят, и довольно быстро. На создание простого кодогенератора у автора ушел примерно час, это время сопоставимо с написанием и отладкой приблизительно 20 хранимых процедур. Соответственно, если их больше, то затраты на создание кодогенерирующего инструмента окупятся сразу же, к тому же в случае изменений в структуре данных код не придется править вручную – просто запустить кодогенератор еще раз, и он построит все процедуры заново с использованием новых метаданных [1, 5]. Этот процесс называется активной кодогенерацией [1, 2, 6], в результате которой получается код, не требующий никакого дополнительного вмешательства со стороны разработчика. В отличие от активной кодогенерации, результатом пассивной кодогенерации обычно является код, состоящий из пустых деклараций и синтаксических конструкций, которые разработчик расширяет по своему усмотрению. Пример удачного применения пассивной кодогенерации – среды визуальной разработки ПО, например, Microsoft Visual Studio, Borland Delphi/C++ Builder. Пассивный кодогенератор может быть удачно применен для построения метаданных для активного кодогенератора.) Если же этот инструмент используется на нескольких проектах и поддерживается на уровне большой организации, то его полезность трудно переоценить.

Выводы

При определенных условиях создание и использование кодогенерирующих инструментальных средств позволяет существенно улучшить процесс разработки ПО, избавить разработчика от рутинного кодирования и отлаживания рутинного кода. Такими условиями являются: необходимость создания большого объема кода, наличие времени на разработку и отладку собственно самого кодогенерирующего приложения, поддержка кодогенерирующего инструментария на корпоративном уровне и повторное его использование в разных проектах. Интроспекция является эффективным способом избавления от необходимости ручного сбора метаданных для кодогенерации, она может быть использована не только для извлечения метаданных из БД, находящейся под управлением СУБД, предоставляющей для этого сервисы, но и для интроспекции откомпилированных библиотек кода. Разумеется, это касается только платформ работающих с управляемым кодом, таким как, например, Microsoft .NET или Java.

В дальнейшем возможно появление систем, генерирующих код приложений полностью на основании некоторых формализованных требований, и собственно ручное кодирование из процесса разработки будет исключено вообще.

Литература

1. Hunt A., Thomas D. The Pragmatic Programmer – from Jour-neyman to Master. – Addison-Wesley, 1999. – 352 p.

2. Beck K., Fowler M. Extreme Programming Explained. – O'Reilly, 2001. – 190 p.

3. Корявченко А. Алгоритмы кодогенерации. – [Электрон. ресурс]. – Режим доступа: http://www.rsdn.ru/article/dotnet/codegen.xml.

4. Code generation.network. – [Электрон. ресурс]. – Режим доступа: www.codegeneration.net.

5. Code generation in Microsoft .NET, Kathleen Dollard. – APress, 2004. – 730 p.

6. Jack Herrington. Code generation in Action // Computers. – 2003. – № 2. – С. 35 -39.

Поступила в редакцию 15.05.2006

Рецензент: д-р техн. наук, проф. Г.Н. Жолткевич, Харьковский национальный университет им. В.Н. Каразина, Харьков.