第八章 字符串模板

一般意义上编译器开发中最终要生成二进制代码。java 和 C#语言编译器会生成相对独立的字节码。实现开发的情形很多，除了开发编译器以外，有时还要开发分析器和翻译器。这需要将一形式的文本翻译成另一种形式的文本输出。例如一个将 java 代码转换成 C#的代码的工具要在解析 java 语法之后要生成 C#的代码。这时我们需要编写大量生成字符串的代码。ANTLR 提供了 String Template 帮助实现生成字符串的操作。

有些编程经验的人都知道，生成字符串时如果使用“+”、“Append”这种硬编码的方式会影响程序的可扩展性，也不容易修改，因为这种代码可读性比较差，代码段变得很复杂。使用 String Template 可以很好地解决这个问题。也比较容易实现。

String Template 是一种字符串的生成工具。它不但可以在 ANTLR 开发中帮助生成代码，它本身也可以作为单独的工具来使用。String Template 可以在 java、C#、C 等开发中使用。

String Template 官方网站是 http://www.stringtemplate.org/。

8.1 文本生成

你在开发时生成文本字符串时怎么做呢？是使用“+”还是用“Append”？我们知道 Append 比使用“+”好，因为“+”会生成很多临时对象而影响程序的执行效率。但是有经验的程序员知道这两种方法都不好，把字符串以硬编码的方式完全写在程序中，不但编码工作量很大，也不容易修改。

```
string str = "SELECT " + fieldName + " FROM " + tableName;
```

这句代码中组成 SELECT 语句中有两处是变化的，查询的字段和表。如果还是这两个变量我们改变一下字符串"SELECT Top 1 " + fieldName + " FROM " + tableName。这时不得不重新编译程序。如果我们把字符串标记出可变部分 SELECT Top1 {0} FROM {1}并保存到文件中，然后我们在程序执行时读入这个字符串并用参数来替换{0}和{1}处的字符串。下面是一个 C#的代码示例。

```
StreamReader streamReader = new StreamReader("temp.txt");
string strS = String.Format(streamReader.ReadLine(), fieldName,
                            tableName);
```

这样我们再修改字符串时只需修改 temp.txt 中的内容就可以了，无需重新编译程序，程序负债只传递参数就可以了。

8.2 介绍 String Template

String Template 是一个独立的工具。String Template 并不是一个很简单的工具，它包含很多内容。所以这一节我们首先全面地学习一下 String Template 这个工具的使用方法，对 String Template 有全面的了解之后再学习它如何和 ANTLR 结合使用。这一节的程序示例也只针对 String Template 与 ANTLR 无关。

String Template 是将输出的字符串形式定义成一种模板形式，然后通过这些模板加上传入的参数据来生成字符串。下面我们先来开发一个最简单的示例。

```
Java:
import org.antlr.stringtemplate.*;
import org.antlr.stringtemplate.language.*;
...
public class testST
{
    public static void main(String[] args) throws Exception
    {
        StringTemplate hello = new StringTemplate("Hello, $name$");
        hello.setAttribute("name", "World");
        System.out.println(hello.toString());
    }
}
C#:
using Antlr.StringTemplate;
static void Main()
{
    StringTemplate hello = new StringTemplate("Hello, $name$");
    uery.SetAttribute("name", "World");
    Console.Out.WriteLine(hello.ToString());
}
```

在这个示例中首先创建了一个 StringTemplate 对象，StringTemplate 对象对应一个字符串模板。构造函数传递了一个字符串参数"Hello, $name$"，"Hello, $name$"是一个很简单的字符串模板，其中$name$代表一个参数它是可替换的部分。接下来调用 setAttribute 方法并用两个参数来指明$name$部分使用“World”来替换。此示例最后输出“Hello World”。

8.2.1 文件模板

String Template 可以将字符串模板以文件的形式保存，例如前一个示例中的"Hello, $name$"存放到 HelloWorld.st 文件中。这样这个字符串模板就有了一个名字叫 HelloWorld 其内容在 HelloWorld.st 文件中。在使用文件模板的情况下调用程序有些不同。请看下面的示例程序：

```
C#
StringTemplateGroup group = new StringTemplateGroup("myGroup");
StringTemplate query = group.GetInstanceOf("HelloWorld");
query.SetAttribute("name", "world");
Console.Out.WriteLine(query.ToString());
```

程序中首先创建 StringTemplateGroup 类而不是 StringTemplate 类。StringTemplateGroup 类代表一系列模板。它构造函数中传递一个模板组名，我们可以任意为模板组起一个名字如“ myGroup”。创建了 StringTemplateGroup 对象后调用 GetInstanceOf 方法，这个方法的参数是字符串模板名字。如 GetInstanceOf("HelloWorld")会加载 HelloWorld.st 文件中的模板。然后和前面的示例一样用 SetAttribute 传递参数并输出结果。

8.2.2 模板组

生成复杂字符串时一般要用多个模板来完成，如果把第一个模板都分开放到不同的文件中会比较麻烦。所以 StringTemplate 加入了模板组的概念，可以将一系列的模板存放到一起。由于多个模板放在一起，多个模板之间就要互相区分，所以 String Template 为模板组定义了一种新的格式。下面我们定义一个简单的模板组。

```
group H;
<! 这是一个简单的模板组 !>
Hello(name) ::= "Hello, <name>"
Hello2(name) ::= <<
    你好, <name>
>>
```

模板组首先要以 group 加模板组的名称开头后面以“;”结束。由于有多个模板，所以要为第一个模板起一个名字以区别多个模板。每一个模板都以一个模板名开始，后面用类似于函数参数定义的方式把模板中需要替换的参数列在括号中，如 Hello(name) 就是一个模板头。在模板头后面使用“::=”分隔后面为模板体。，如果模板是单行则使用“”来定义模板的范围。如果模板是多行则使用<<>>来定义模板的范围。模板中也可以加入注释<!!>之中的内容是注释内容。

还需要注意的是在多行模板中如果第一行和最后一行是空行则这种空行会被忽略掉。在模板组文件中可替换部分参数是用<>来表示而不是$$。稍后我们会知道如何选择这两种符号。另外与文件模板一样模板组保存文件的文件名要和模板组的名字相同。我们把上面的模板组保存为 H.stg。程序中加载使用模板组的代码有些不同，有两种方法加载模板组，下面看一下程序代码：

方法一：C#

```
StringTemplateGroup coreTemplates =
    new StringTemplateGroup(new StreamReader("H.stg"),
            typeof(Antlr.StringTemplate.Language.DefaultTemplateLexer));
StringTemplate t = coreTemplates.GetInstanceOf("Hello");
```

第一种方法使用 StringTemplateGroup 类来加载模板组文件。构造函数的第一个参数

是一个 StreadReader 对象来指定模板组的文件名。

我们再看一看加载模板组的方法二：

```
IStringTemplateGroupLoader loader = new
    CommonGroupLoader(ConsoleErrorListener.DefaultConsoleListener,
     "temp");
StringTemplateGroup.RegisterGroupLoader(loader);
StringTemplateGroup tempGroup = StringTemplateGroup.LoadGroup("H");
StringTemplate temp = group.GetInstanceOf("Hello");
```

方法二使用 CommonGroupLoader 类来加载模板组文件，CommonGroupLoader 可以加载一个目录下的所有模板文件。它的第二个构造函数用来指定路径。创建 CommonGroupLoader 类对象后使用 StringTemplateGroup 的 RegisterGroupLoader 方法与 StringTemplate Group 对象相关联。然后调用 StringTemplateGroup 类的 LoadGroup 方法并传入使用模板组的名称。

GroupLoader 文法将 CommonGroupLoader 类传递给 StringTemplateGroup 类。StringTemplateGroup 的 LoadGroup 方法用来选择当前要使用哪个模板文件。LoadGroup 方法返回 StringTemplate 对象。到此我们介绍了三种加载模板的方法 StringTemplate、StringTemplateGroup 和 GroupLoader。

8.2.3 参数符号

8.2.2 方法一中 StringTemplateGroup 的构造函数的第二个参数用来指示这个模板组使用$$来表示可替换参数。由于在模板中默认的参数标识符为$$，模板组默认为<>，所以想改变默认的符号时就要用第二个参数 typeof(DefaultTemplateLexer)来指定，在 java 程序中则用 AngleBracketTemplateLexer.class 的写法，请看下表指示出了模板和模板组默认情况下的参数标识符号和符号对应的类型：

	默认	对应类型
模板	$$	DefaultTemplateLexer
模板组	<>	AngleBracketTemplateLexer

表 8.1

String Template 中定义这两种参数的符号的目的是为了避免与字符中的其它字符混淆。在 html 和 XML 中如果用<>会很容易混淆所以这时用$$比较好。在 SQL 中则是使用<>比较好。

8.2.4 模板使用详细介绍

8.2.4.1 集合参数

StringTemplate 中的参数可以重复设置，以"Hello, <name>"为例，可以用.SetAttribute("name", "world");设置参数一次。也可多次重复地设置 name 参数。

```
temp.SetAttribute("name", "li");
temp.SetAttribute("name", "zhang");
temp.SetAttribute("name", "zhao");
```

如果多次设置参数，此参数就成为了集合。上面的代码使 name 参数成为一个有三个元素 List<string>。也可以直接使用 List 对象来设置集合参数。请看下面的代码：

```
List<string> nameList = new List<string>();
nameList.Add("li");
nameList.Add("zhang");
nameList.Add("zhao");
temp.SetAttribute("name", nameList);
```

8.2.4.2 构造集合

我们可以将多个值构造成一个集合参数。这样就可以按照集合参数来处理。构造集合时可以使用["A","B","C"]的格式。注意["A","B","C"]是做为一个参数来用的。这与 List<string> strList = new List<string>; strList.Add("A"); strList.Add("B"); strList.Add("C");的 strList 是等价的。下面看一个构造集合的例子：

```
StringTemplateGroup coreTemplates =
            new StringTemplateGroup(new StreamReader("t.stg"));
StringTemplate t = coreTemplates.GetInstanceOf("ListCon");
t.SetAttribute("x", "X1");
t.SetAttribute("y", "Y1");
t.SetAttribute("z", "Z1");
Console.Out.WriteLine(t);
```

t.stg：

```
group t;
ListCon(x,y,z) ::= "$[x, y, z]:{it | $it$ }$";
```

运行后输出：X1 Y1 Z1。ListCon 模板中的:{|..}的写法是内嵌模板，作用是枚举集合中的每个元素并对每一个元素进行处理。内嵌模板的格式为 <list:{item | ... item...}>，{}中为内嵌模板体，item 代表集合中的某一个元素。| 符号前面的是变量列表，| 后面为输出的格式定义。

8.2.4.3 集合函数

String Template 中有几个对集合操作实用函数，分别是 first, rest, last, length, strip。我们在操作集合时有时对于第一个元素或最后一个元素做的处理和其它元素可能会不同。例如我们生成一段连接字段列表的字符串的代码。字段列表之间用“，”分隔。

```
string strSQL = "";
<fields:{ f | strSQL += \"<f>,\";}>
```

上面的模板的写法要在<fields>之前先定义 strSQL，并且在最后一个字段连接后会多余一个“，”。下现我们来利用 first 和 rest 函数来实现上面的示例。

<first(fields):{ f | string strSQL = \"<f>\";}>
<rest(fields):{ f | strSQL += \",<f>\";}>

first 函数的用法与前面介绍的内嵌模板相同只是其 f 变量只对 fields 集合中的第一个元素操作。这样我们可以将生成的第一条语句与定义变量写在一起，并且“，”间隔符问题也解决了。

last 函数和 first 函数相对用来对集合的最后一个元素进行操作。lenght 函数返回一个集合的个数。

8.2.4.4 参数的附加选项

参数在 String Template 里是最重要的部分。参数定义在“$$”和“<>”定义符中，定义符中还可以添加一些附加选项来辅助定义参数的格式。参数的附加选项有三项：separator 用来定义集合元素之间的分隔符。wrap 用来定义参数是否自动换行。null 用来定义如果参数为空时用什么字符串来代替。下面我们用一个示例来学习一下这三个选项的用法。

设置分隔符：
INSERT INTO T1 VALUES (<fields; separator=", ">)
st.setAttribute("fields", "1");
st.setAttribute("fields", "2");
st.setAttribute("fields", "3");
如果如前面讲过的示例三次设置 name 参数，输出为：INSERT INTO T1 VALUES (1,2,3)。

设置自动换行：

Hello, <name; wrap >

Java 代码示例:

```
StringTemplate st = new StringTemplate("<name;wrap>",
                        AngleBracketTemplateLexer.class);
st.setAttribute("name ", "a");
st.setAttribute("name ", "b");
st.setAttribute("name ", "c");
st.setAttribute("name ", "c");
st.setAttribute("name ", "e");
st.setAttribute("name ", "f");
st.setAttribute("name ", "g");
st.setAttribute("name ", "h");
st.setAttribute("name ", "i");
System.out.println(st.toString(5));
```

输出为：abcef

ghi

设置空字符：

INSERT INTO T1 VALUES (<fields; separator=", ", null="0">)

StringTemplate st = new StringTemplate("INSERT INTO T1 VALUES (<fields; separator=", ", null=\"0\">", AngleBracketTemplateLexer.class);

ArrayList al = new ArrayList();

al.add("1");

al.add("2");

al.add(null);

al.add("4");

st.setAttribute("fields", al);

System.out.println(st.toString(5));

输出为：INSERT INTO T1 VALUES (1,2,0,4)。结果中 String Template 使用"0"代替 null 参数。

8.2.4.5 对象参数

String Template 可以使用对象作为参数来传递复杂的信息到字符串模板中，对象可以利用属性来传递多个值这在实际应用中使用的很广泛。对象参数作为对象在访问它的属性时使用参数名加属性名“param.property”的形式。下面我们看一个示例：

定义如下文法：

ListCon(p) ::= "Name: <p.name>, Age：<p.Age>"

在程序中定义一个 Person 类具有 Name 和 Age 两个字段，并将 Person 的对象传递给参数 p。

Person p = new Person();

p.Name = "gao";

p.Age = 29;

temp.SetAttribute("p", p);

有时属性的名称可能会和相应语言的关键字或语法冲突。这时可以用别一种访问对象属性的方法。它的写法为：param.("propertyName")

ListCon(p) ::= "Name: <p.("name">, Age：<p.("Age")>"

有时我们需要让对象的属性在运行时可变，这时我们可以将属性名定义成变量。它的写法为：param.(propertyNameVar)

ListCon(p, proName) ::= "Person Property: <p.(proName)> "

对应的代码为：

temp.SetAttribute("p", p);

temp.SetAttribute("proName", "Name");

8.2.4.5 字典数据类型的访问

String Template 允许类似于访问对象参数一样来访问字典数据类型。在模板中参数

的写法与对象参数一样。如$user.name$。然后把字典对象传入，字典对象名要定义成user 而 name 是字典里的键值。这样$user.name$就会用 user 字典里 name 对应的值来替换。下面看一下程序示例：

```
Java:
StringTemplate a = new StringTemplate("$user.name$, $user.phone$");
HashMap user = new HashMap();
user.put("name", "Terence");
user.put("phone", "none-of-your-business");
a.setAttribute("user", user);
System.out.println(a.toString());
C#:
StringTemplate a = new StringTemplate("$user.name$, $user.phone$");
Dictionary<string, string> user = new Dictionary<string, string>();
user.Add("name", " Terence");
user.Add("phone", " none-of-your-business");
a.SetAttribute("user", user);
Console.Out.WriteLine(a.ToString());
```

8.2.4.6 模板调用

我们可以从一个模板中调用别一个模板就象调用函数那样。这样可以让不同的模板完成不同的工作。我们先说一下存在于.st 文件中的模板。.st 文中只有一个模板，这时模板只有内容没有名字和参数表，所以调用它时只能使用模板的文件名，比如一个模板保存在temp.st 模板文件中调用这它的时候要写成$temp()$。在调用模板时 String Template 会寻找与模板名同名的.st 文件，如查找不到这个文件会出现异常。这时大家可能还有疑问就是我们怎么传递参数呢？这很简单例如果模板里有 param 参数它的写法是$temp(param="abc")$，

对于模板组中的模板调用（也就是存在于.stg 文件中的模板）就更简单了。因为模板组中的模板都有名字和参数表，所以调用时不用指定参数名。如写成$temp("abc")$就可以。大家是否还记得我们前面讲过的模板组加载方法的方法二，其中使用 CommonGroupLoader 类进行模板的加载。这种方法可以让 String Template 对一个目录下的所有模板组起作用，这时当一个模板组里的模板调用另一个模板组里的模板时是允许的。

下面看一下程序示例

```
CREATE(tableName, Fields) ::= <<
CREATE TABLE <tableName> (
    <Fields:{f | <FieldP(f)>}; separator=",\r\n">
    )
>>
FieldP(f) ::= <<
    <f.Name> <f.mType>
>>
```

这是一个产生 Create Table 语句的模板。它有三个参数 tableName 是表名，Fields

是一个对象列表，类型为 List<Field>。Field 类是一个表示字段信息的类。它有一个 Name 属性代表字段名和一个 Type 属性代表字段类型。下面是 Field 类的代码：

```
public class Field
{
    public Field(string Name, string mType)
    {
        this.Name = Name; this.mType = Type;
  }
  public string Name;
  public string Type;
}
```

在 CREATE 模板中循环 Fields 列表并调用了 FieldP 模板将第一个代表字段信息的 Field 对象做为参数传递给 FieldP 模板来生成关于字段的内容。

8.2.4.7 条件参数

有时需要根据某些情况选择性地输出某些内容，String Template 中有一种条件参数。可以用布尔变量来选择性地输出内容。这种条件参数的的写法是<if(boolVar)>相当于 if 语句。下面我们看一个程序示例：

```
group Select
SelectSQL(fields, Table, isWhere) ::= <<
SELECT <fields; separator=", "> FROM <Table>
<if(isWhere)>
    WHERE Name = 'a'
<else>

<endif>
>>
```

这个示例中 SelectSQL 的 isWhere 参数为真时会输出 WHERE Name='a'字符串。条件变量 isWhere 为假时会输出<else>的内容，<endif>用来标识<if>的结束。

```
f.SetAttribute("fields", new Field("ID", "int"));
f.SetAttribute("fields", new Field("Name", "string"));
f.SetAttribute("fields", new Field("AddTime", "DateTime"));
f.SetAttribute("fields", new Field("AddUser", "int"));
f.SetAttribute("Table", "Table1");
f.SetAttribute("isWhere", true);
Console.Out.WriteLine(f.ToString());
```

这段代码建立了 ID、Name、AddTime 和 AddUser 这个字段，表名为 Table1，对应 isWhere 参数要传递布尔变量。我们可以得到 SELECT ID,Name,AddTime,AddUser FROM Table1 WHERE Name = 'a'字符串。

8.2.4.8 模板组的继承

String Template 还支持一些更加高级的功能，一个模板组可以继承别一个模板组，

这与面向对象中的继承一样。我们可以合理的重用已有的模板，在已有模板的基础上创造新模板。下面看一下如何实现模板组的继承。

```
group Java;
Class(className, Methods) ::= <<
   public class <className> {
         <Methods:{m | <Method(m)>}; separator="\r\n">
   }
>>
Method(m) ::= <<
    public <m.mType> <m.Name>() {
       return null;
    }
>>

group CSharp : Java;
Class(className, Methods, Properties) ::= <<
   public class <className> {
         <Properties:{p | <Property(p)>}; separator="\r\n">
         <Methods:{m | <Method(m)>}; separator="\r\n">
   }
>>
Property(m) ::= <<
    private <m.mType> _<m.Name>;
    public <m.mType> <m.Name>() {
       get {
            return _<m.Name>;
       }
       set {
            _<m.Name> = value;
       }
    }
>>
```

下面是运行代码：

```
IStringTemplateGroupLoader loader = new CommonGroupLoader
            (ConsoleErrorListener.DefaultConsoleListener, "temps");
StringTemplateGroup.RegisterGroupLoader(loader);
StringTemplateGroup coreTemplates =
StringTemplateGroup.LoadGroup("CSharp");
StringTemplate t = coreTemplates.GetInstanceOf("Class");
t.SetAttribute("className", "Class1");
t.SetAttribute("Methods", new Method("getClassName", "string"));
t.SetAttribute("Methods", new Method("Save", "void"));
t.SetAttribute("Properties", new Property("Name", "string"));
```

```
t.SetAttribute("Properties", new Property("Tag", "object"));
Console.Out.WriteLine(t);
```

8.2.4.9 模板组接口

除了继承 String Template 还支持模板组接口，模板组接口与面向对象中的接口是一样的概念。我们可以为某种设计定义一组模板名称和参数而不去实现模板内容，以后继承模板组接口的模板组一定要按照这个设计去实现，这样模板接口起到了制定标准的作用。

```
interface IJava;
Class(className, Methods);
Method(m);

group Java implements IJava;
Class(className, Methods) ::= <<
   public class <className> {
         <Methods:{m | <Method(m)>}; separator="\r\n">
         <foreach m in Methods>
             <Method(m)>
         <end>
   }
>>
Method(m) ::= <<
     public <m.mType> <m.Name>() {
        return null;
     }
>>
```

8.3 ANTLR 中使用 String Template

对 String Template 有了全面了解之后我们回到 ANTLR 中，我们来学习一下如何将 String Template 和 ANTLR 结合使用，这是本章的关键。在 ANTLR 文法中有一个 output 选项这个选项我们已经学过并且经常使用。output=AST 用来生成语法树，output 选项还可以设置成 ouput=template 这样设置代表规则将输出字符串模板。ANTLR 文法规则中加入以–> templateName (paramName=$paramValue,...)的形式叫做模板重写规则。加入到规则的后面，与 String Template 结合起来。

8.3.1 简单的示例

下面看一个示例，这个示例实现根据配置文件的信息来生成代码的功能。首先新建一个模板组 Test.stg。

```
group TestTS;
Test(ctl1, ctl2, ctl3) ::= <<
```

```
if(<ctl1>.Checked) <ctl2>.Enabled = true; else <ctl3>.Enabled = false;
>>
```

我们定义了一个非常简单的模板组，其中有一个 Test 模板定义了三个参数，这三个参数分别代表三个控件的名字，然后生成一段与这个三个控件有关的代码。这样的代码可能我们在开发中经常书写，我们的目的是用简单代替复杂。下面我们再新建一个.g 文件：

```
grammar TestTS;
options {
  output=template;
  language=CSharp;
}
checkControl : a=ID '>' b=ID ',' c=ID ';'
 -> Test(ctl1={$a.text}, ctl2={$b.text}, ctl3={$c.text});
ID: ('A'..'Z' | 'a'..'z') + ;
WS :(' ' |'\t' |'\n' |'\r' ) {Skip();} ;
```

这个文法定义了一个 a > b, c;形式的规则 checkControl，我们现在来做一个简单的代码转换工具，a、b 和 c 分别是三个控件的名字，a 控件是一个 CheckBox 或 RadioBox 控件如果它的 Checked 属性为 true 则把 b 控件的 Enabled 属性设为真，如果 Checked 属性为 false 则把 c 控件的 Enabled 属性设为真。ID 规则为了简单只能用字母表示不能用数字请大家注意。在文法中我们要注意的是规则后半部分与 String Template 有关系，它调用了上面我们定义的 Test 模板并将 a、b、c 三个符号的内容作为参数传递给 Test 模板，现在根据这个文法生成分析器后，我们来编写运行代码：

```
IStringTemplateGroupLoader loader =
                                                                new
CommonGroupLoader(ConsoleErrorListener.DefaultConsoleListener,
    "C:\Templates");
StringTemplateGroup.RegisterGroupLoader(loader);
StringTemplateGroup                coreTemplates                =
StringTemplateGroup.LoadGroup("TestTS");
TestTSLexer lexer = new TestTSLexer(new ANTLRFileStream("TestTS.txt"));
CommonTokenStream tokens = new CommonTokenStream(lexer);
TestTSParser parser = new TestTSParser(tokens);
parser.TemplateLib = coreTemplates;
TestTSParser.checkControl_return r = parser.checkControl();
Console.Out.WriteLine(r.st.ToString());//使用返回对象的 st 属性，有些版本的 antlr 用 ST（大写）。
```

运行代码是一段结合了 StringTemplate 和 ANTLR 的代码，前半部分使用 CommonGroup Loader 加载模板组，生成模板组对象 coreTemplates 然后将其设置给 parser 的 TemplateLib 属性，这样我们把 ANTLR 的语法分析器与 StringTemplte 组装在一起，文法中的->template Name(paramName=paramValue,..)形成就可以找到对应的模板并生成字符串。

文法中 output 设置项为 template 所以规则生成的不是语法树而是字符串模板 String Template，可以使用返回类型的 st 属性来获得并调用 ToString()方法来获得字

符串。下面是规则返回类型的定义。类中定义了 StringTemplate 类型的 st 属性，这是 output 设置项为 template 的结果。

```
public class checkControl_return : ParserRuleReturnScope
{
    public StringTemplate st;
    public override object Template {
       get { return st; }
    }
   public override string ToString() {
         return (st == null) ? null : st.ToString();
    }
};
```

下面我们用生成的分析器来分析下面的输入字符串

radIsExist > txtUserName, txtRoleName;

分析器会输出：

if(radIsExist.Checked) txtUserName.Enabled = true; else txtRoleName.Enabled = false;

8.3.2 集合参数的传递

我们修改一下前面的示例，让我们的分析器可以分析 a > b1 b2... | c1 c2...字符串。这次我们要根据 a 控件的 Checked 属性决定两组控件的 Enabled 属性设置。这需要给 String Template 传递集合参数。下面看一下修改后模板组文件：

```
group TestTS2;
Test(ctl1, ctl2List, ctl3List) ::= <<
if(<ctl1>.Checked)
    <ctl2List:{p | <p.Text>.Enabled = true;}>
else
    <ctl3List:{p | <p.Text>.Enabled = false;}>
>>
```

原来的 ctl2,ctl3 参数改为 ctl2List 和 ctl3List 它们变成了集合参数，所以使用内嵌模板来循环这两个集合参数，来生成两组语句。下面我们再看一下新的文法内容：

```
grammar TestTS2;
options {
  output=template;
  language=CSharp;
}
checkControl : a=ID '>' b+=ID+ ',' c+=ID+ ';' -> Test(ctl1={$a.text}, ctl2List={$b}, ctl3List={$c});
ID: ('A'..'Z' | 'a'..'z') + ;
```

```
WS :(' ' |'\t' |'\n' |'\r' ) {Skip();} ;
```

这个修改后的文法中变量 b 和 c 通过使用+=操作符变成了集合类型这在前面的章节讲过。然后再将集合 b 和 c 传递给 ctl2List 和 ctl3List 参数。这样就很简单地实现了集合参数的传递。生成代码编译运行示例并对下面的字符串进行分析：

```
radIsExist > txtUserName txtUserCompany txtUserNo, txtRoleName txtRoleNo txtRoleFunc;
```

分析器输出：

```
if(radIsExist.Checked)
    txtUserName.Enabled = true;txtUserCompany.Enabled = true;
    txtUserNo.Enabled = true;
else
    txtRoleName.Enabled = false;txtRoleNo.Enabled = false;
    txtRoleFunc.Enabled = false;
```

8.3.3 在 Actions 中调用模板

Actions 在文法中起着很重要的作用，我们可能在 Actions 中创建新的对象。这时如果使用 String Template 就涉及到在 Actions 中调用模板，在 Actions 中调用模板与前面使用模板重写规则的写法不同。下面我们学习一下如何在嵌入文法的 Actions 中调用模板。

```
① checkControl : a=ID '>' b=ID ',' c=ID ';'
   -> Test(ctl1={$a.text}, ctl2={$b.text}, ctl3={$c.text});
② checkControl : a=ID '>' b=ID ',' c=ID ';'
   {$st = %Text(ctl1={$a.text},ctl2={$b.text}, ctl3={$c.text});};
③ checkControl : a=ID '>' b=ID ',' c=ID ';'
   {$st = %Text();
    %{$st}.ctl1=$a.text;
    %{$st}.ctl2=$b.text;
    %{$st}.ctl3=$c.text;
   };
```

这三种对模板的调用是作用完全相同的三种写法，②和③是两种在 Actions 中调用模板的写法。在 Actions 表示模板要在模板名前面加入%符号，②与③的不同在于②在调用模板的时候直接传入了参数。而③是先生成模板对象并赋给规则的 st 属性然后再设置模板的 x 和 y 的两个属性。①、②与③在分析器中生成的代码如下：

```
① 和② ：
retval.st = templateLib.GetInstanceOf("Test",
new STAttrMap().Add("ctl1", a.Text).Add("ctl2", b.Text).Add("ctl3", c.Text));
```

① 和②生成代码是将各个参数组成一个 STAttrMap 类型的集合，然后通用 GetInstanceOf 方法的第二个参数传递给模板。这种写法我们以前没有用过，也是生成模板的一种方法。

③：

```
retval.st =  templateLib.GetInstanceOf("Text");
(retval.st).SetAttribute("ctl1",a.Text);
(retval.st).SetAttribute("ctl2",b.Text);
(retval.st).SetAttribute("ctl3",c.Text);
```

③ 生成的代码与我们前面学到的写法是一样的前建立模板对象然后再用 SetAttribute 方法设置参数。

我们还有必要对③中的%{$st}.ctl1=$a.text;写法做一些解释。其实如果我们知道 $st 在生成代码后是 retval.st，那么我们可以把这一句写成 retval.st.ctl1=$a.text。这就是说如果在 Actions 是引用的是以$开头的表达式。那么如果想引它的属性就要写成%{$st}.的形式。%{}可以把 Antlr 变量转换成正确的生成代码之后的形式。如果不这样比如写成$st.ctl1=$a.text;那么生成的代码为(st).ctl1=a.Text 这样显然是错误的。这就是要写成%{$st}的原因。

野猪一样帅(65557557) 2009-04-25 23:55:47

stringtemplate 的问题。一个文法怎么对应多个 st 模板， 目前好像是 c.g => 一个 c.stg?

野猪一样帅(65557557) 2009-04-25 23:56:26

如果 c.stg 没有用继承其他 stg 模板的话。

Alien~(37287019) 09:36:46

为什么只能一个 c.stg?

野猪一样帅(65557557) 09:38:58

嗯，我理解错了。

野猪一样帅(65557557) 09:39:24

不是.st 的文件可以有多个

野猪一样帅(65557557) 09:39:30

.stg 的也可以多个吗?

Alien~(37287019) 09:40:57

可以 parser.TemplateLib = coreTemplates;