RE 控制程序源代码自动生成程序的研究与实现①

彭四伟 陈玉萍 (北京化工大学 信息科学与技术学院 北京 100029)

摘 要： 讨论了一个基于 Orad Render Engine 的 RE 控制程序源代码自动生成程序的设计思想与实现技术，通过编写一个文档分析程序，以控制命令的说明文档作为输入，按照预定的正则表达式规则，分析出命令的结构化数据，并动态创建 XML 格式的代码模板。最终系统根据动态生成的模板自动生成出控制程序的源代码。

关键词： 渲染引擎；源代码自动生成；正则表达式；结构化数据；XML 模板

Research and Implementation of Render Engine Controller's Auto Source Code Generator

PENG Si-Wei, CHEN Yu-Ping

(College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: This paper analyzes the design and realization technology of auto source code generator based on Orad render engine controller. It uses a document analysis process, which takes a documentation of control commands as input and conformed to predetermined regular expression rules, to analyze the structured data of commands, and to dynamically generate XML-based code templates. Finally, this system automatically generates the source code of controller in accordance with the generated templates.

Keywords: render engine; auto source code generation; regular expressions; structured data; XML-based templates

1 引言

随着计算机技术的快速发展与日益成熟，当今社会对软件产品的需求也一直处于增长的态势，对软件质量的要求也不断提高。要满足巨大的软件需求，同时又要保证软件质量、提高软件的生产效率、降低软件的开发周期和成本，仅依靠程序员手工编码不仅要耗费大量的时间和精力，而且还存在许多不稳定、难以预料的因素，造成程序的不可靠[1]，因此，软件工程领域提出了源代码自动化生成的软件开发方法[2]。源代码自动生成技术可以很好地完成重复性代码的自动生成，减轻程序员的编码负担，提高代码可读性[3]，降低软件开发的周期和成本，快速生成并加入新的组件以使软件易于维护和扩展；从而可以提高软件的生产效率，提高软件系统的健壮性、稳定性、可扩展性、可维护性以及可操作性[4]。

为了对源代码自动生成技术进行研究，本文提出编写一个依据 Orad Render Engine(简称 RE)控制命令文档来自动生成 RE 控制程序源代码的源代码自动生成程序。该自动生成程序能够分析命令协议，并能够解决依靠手工编码无法应对的 RE 升级频繁、命令数量繁多以及命令协议变化等一系列问题，从而极大地减少了编程人员的工作量，提高了工作效率，使编程人员能够把精力更多地集中在创造性的开发上[5]，也有效地保证了程序的动态性和扩展性[6]。

2 系统设计

本系统的设计源于一个渲染引擎即 RE 的控制系统，该控制系统通过向 RE 的控制端口发送控制命令来控制 RE 的运行。RE 会提供一个关于控制命令的文档，该文档中包含很多控制命令的说明，而且每条命令的格式稍有不同。但总体而言，该说明文档是具有一些格式规律的，而且每条控制命令也遵循一定的格式规律。本系统就是根据这个文档来生成 RE 控制程序源代码的。

本系统以控制命令的说明文档作为输入，按照预

① 收稿时间:2009-07-26;收到修改稿时间:2009-09-02

定的正则表达式规则，从文档中提取出各个控制命令的命令名、命令描述、命令参数说明、命令文本示例和命令反馈文本示例等命令的结构化数据，同时还分析命令的参数结构，最后根据动态分析出的代码模板自动生成控制程序的源代码。整个系统的工作模型如图 1 所示。具体到内部实现，它主要由三部分组成：控制命令说明文档的读入与分析、参数提取与模板生成、代码的自动生成。

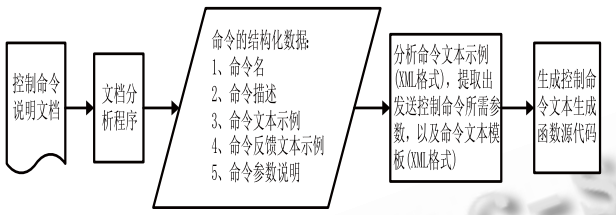

图 1　源代码自动生成系统的工作模型

2.1 分析控制命令说明文档

读入控制命令说明文档，先按照一定的预处理规则表(由匹配规则和替换规则构成一条预处理规则)对文档进行预处理，再按照行类型识别的正则表达式规则分析文档，将说明文档分解为以命令为单位的结构化数据。以说明文档中的“ Get Scene Animations ”命令为例，其在文档中的说明内容如下：

2.4.1Get scene animations

```
<GetIds Id="21">
  <AppItemId>
    <SceneId Name="NewRE/scena2_ new"/>
    <SubItemId ClassType="^BaseAnimation-
Group">
      <Id Name=""/>
    </SubItemId>
  </AppItemId>
</GetIds>
```

Retrieves all animations from the scene.

Parameters description:

-SceneId Name – scene name

Response:

```
<Ids Id="8" Reason="1" RequestId="21"
SourceClientId="6">
  <AppItemId>
    <SceneId Name="NewRE/scena2_new"/>
    <SubItemId ClassType="^BaseAnimation-
Group">
      <Id Name=""/>
    </SubItemId>
  </AppItemId>
  <Ids Count="3">
    <Item ClassType="^AnimationGroup">
      <Id Name="in"/>
    </Item>
    <Item ClassType="^AnimationGroup">
      <Id Name="out"/>
    </Item>
    <Item ClassType="^AnimationGroup">
      <Id Name="Main"/>
    </Item>
  </Ids>
</Ids>
<Reply Id="9" Reason="1" RequestId="21"
SourceClientId="6">
  <Error Code="0" Description="Ok"/>
  <Failures Count="0"/>
</Reply>
```

由上面的例子可看出，控制命令说明文档中对于命令的说明主要由五个部分组成：即命令名(也就是以一个标题序号作为开始的独立行，当出现多级标题的情形时，以最后一级标题为命令名)、命令文本示例、命令描述、命令参数说明、命令反馈文本示例。本系统将命令的这些结构化数据作为属性封装在 RECommandStructure 类中。具体描述如下：

1) RECommandName：命令名称

2) RECommandSample：命令文本示例

3) RECommandDescription：命令描述

4) RECommandParam：命令参数说明

5) RECommandResponse：命令反馈文本示例

6) RECommandText：命令文本

7) RECommandCode：命令的文本生成函数代码

其中，前 5 个属性可以直接分析说明文档得出，而命令文本 RECommandText 和命令的文本生成函数代码 RECommandCode 属性需要对说明文档中的示例进行进一步分析转换才能得到。

因为说明文档中对于每个命令的描述都遵循一定的格式，因此，本系统采用正则表达式的方法来提取每个命令的结构化数据。正则表达式的匹配规则如表 1 所示：

表 1 说明文档的匹配规则

匹配行的类型	匹配规则
标题行	@"^\s*(?:\d+(?:\.\d+)*)(.+)\s*$"
XML 文本行	@"^\s*<|>\s*$"
Parameters description:起始行	@"^\s*Parameters description:\s*$"
Response:起始行	@"^\s*Response:\s*$"
其它文本行	@"^(.*)$"

分析说明文档时，以命令为单位，每次读入一行进行分析，当遇到一个标题行时表示一个命令说明的开始，循环匹配直到出现另一个标题行时，表示结束对这个命令的说明。在具体的分析过程中，采用表驱动的方式进行分析。分析驱动表是一个三维表，其中，第一维表示当前的分析状态，第二维表示当前输入行，第三维的值仅为 0 或 1，其中 0 元素表示动作，1 元素表示下一个状态。通过执行指定的动作并转到下一个状态，循环对每一行进行分析。

此外，对分析出的所有 XML 行根据 XML 字符串的根元素不同还需进一步标识出完整的命令文本示例和命令反馈文本示例的 XML 文本段。然后将分析出的所有命令的结构化数据作为解析结果存放在列表容器中，供模板生成、参数提取以及代码生成时使用。

2.2 参数提取与模板生成

在命令文本示例中，含有一些数据常量，如数值、字符串、逻辑值等，这些数据常量有些是命令文本的组成部分，有些则是需要由外部提供的命令参数，所以，还需要对分析出的命令结构化数据中的命令文本示例进行进一步的分析，根据参数规则识别出参数部分。命令文本示例是一段完整的 XML 文本，而且其所有元素都是空元素，即没有以节点文本形式出现的数值，所有的命令数据都被包装在节点属性中[7]，因此，可能的参数都是以 XML 节点属性的形式出现的。在进行参数分析时，先遍历整个命令文本示例的 XML 结构，提取出所有的属性，再按照识别出的参数规则，逐一分析每个属性是否为参数。参数的具体规则为：属性值以特殊符号/、^和@开头的不作为参数；属性值为空的不作为参数；Count 属性、Contents 属性、MethodName 属性、Type 属性不作为参数；以及属性 Name 的属性值等于某些特殊字符串的不作为参数。在具体的分析过程中，将这些规则形式化为正则表达式，排除这些非参数属性，进行参数的识别。这里非参数属性的识别规则的每条规则由两个规则项组成，第一个是属性名识别规则，第二个是属性值识别规则。当一个属性同时满足某一条规则的两个规则项时，则被视为非参数属性。若一个规则项为空则只要满足另一个规则项，则也被视为非参数属性。识别出了非参数属性，则剩下的属性即为参数属性。

另外，还使用了一个类来描述参数的结构信息，该类所定义的属性如下：

1) NodeName：参数所对应的属性所属的节点名

2) AttributeName：参数所对应的属性名

3) ParamType：参数类型

4) ParamName：参数名

这个类结构记录要作为参数的节点属性值对应的节点名、属性名以及与其相对应的参数名及参数类型。以命令为单位，将识别出的每个命令的所有参数记录在列表集合中。

然后，结合识别出的每个命令的参数列表，以及命令所对应的命令文本示例，动态生成 XML 格式的命令文本模板。即模板生成部分主要完成的功能是将已分析出的命令文本示例转换成可供代码自动生成系统进行配置的命令文本模板[8]。其具体流程如图 2 所示。

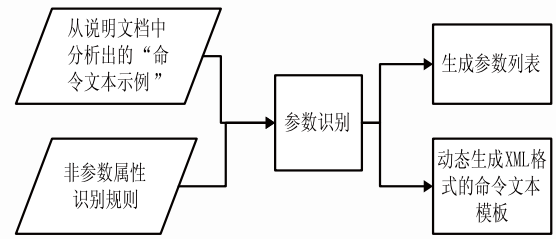

图 2 命令参数列表及命令文本模板的生成过程

为了区分参数与属性值，将需要作为参数的属性值的形式表示为“ __参数的变量名__ ”，其中参数的变量名是以参数属性中的属性名加上‘ a ’构成的。再以命令“ Get Scene Animations ”为例，生成的命令文本模板为：

```
<GetIds Id="__aId__">
  <AppItemId>
    <SceneId Name="__aSceneName__"/>
    <SubItemId    ClassType="^BaseAnimation-
Group">
      <Id Name=""/>
    </SubItemId>
  </AppItemId>
</GetIds>
```

2.3 代码生成

代码生成部分将文本模板通过代码生成器映射为所需要的代码[9]，主要是结合命令结构化数据以及提取出的对应命令的参数列表，将 XML 格式的命令文本

模板转换为命令文本生成函数源代码的过程。因为命令文本实际是一个小型 XML 文档，可以整体看成是一个 XML 节点，因此其代码生成过程的基本框架基本一致，都遵循以下几个步骤，即：创建根节点、创建节点属性以及循环创建子节点。在将文本模板转换成源代码的过程中，需要进行参数与属性值的匹配。其实现通过递归分析命令文本模板，循环遍历整个 XML 结构，将每个 XML 元素及其属性与参数列表中的元素名及其对应的属性名进行匹配，若匹配成功，则用参数名作为该元素对应属性的属性值。仍以命令“ Get Scene Animations ”为例，通过模板及参数匹配，自动生成出的一部分源代码如下：

```
public static string GetSceneAnimations (string aId, string aSceneName)
{
 CommandWriter aCommandWriter = new CommandWriter();
 aCommandWriter.XmlWriter.WriteStartDocument()
 aCommandWriter.XmlWriter.WriteStartElement("GetIds");
 aCommandWriter.XmlWriter.WriteAttributeString("Id",aId);
 aCommandWriter.XmlWriter.WriteStartElement("AppItemId");
 aCommandWriter.XmlWriter.WriteStartElement("SceneId");
 aCommandWriter.XmlWriter.WriteAttributeString("Name",aSceneName);
 aCommandWriter.XmlWriter.WriteEndElement();
 aCommandWriter.XmlWriter.WriteStartElement("SubItemId");
 aCommandWriter.XmlWriter.WriteAttributeString("ClassType",@"^BaseAnimationGroup");
 aCommandWriter.XmlWriter.WriteStartElement("Id");
 aCommandWriter.XmlWriter.WriteAttributeString("Name",@"");
 aCommandWriter.XmlWriter.WriteEndElement();
 aCommandWriter.XmlWriter.WriteEndElement();
 aCommandWriter.XmlWriter.WriteEndElement();
 aCommandWriter.XmlWriter.WriteEndElement();
 aCommandWriter.XmlWriter.WriteEndDocument();
 aCommandWriter.XmlWriter.Flush();
 aCommandWriter.XmlWriter.Close();
 return aCommandWriter.ToString();
}
```

3 结语

本文论述的源代码自动生成程序，可以极大地减少重复代码的编写，使开发人员更加专注于业务逻辑，提高开发效率[10]，具有良好的实用性和可靠性。当然，为了使自动生成的程序更好地符合用户的要求，使系统具有良好的友好性，因此，本系统对自动生成的程序并不排斥手工干预，即允许用户对自动生成代码进行手工修改。针对于此，下一步的研究工作将是继续完善系统功能，引入“ 代码导入 ”和“ 代码对比 ”功能。“ 代码导入 ”功能使用户可以选择要导入的被修改过的代码文件，然后使用“ 代码对比 ”功能，将用户指定的代码文件与自动生成的代码文件进行对比，找到发生修改过的地方，将修改记录下来，当再次生成代码时，可询问用户是否应用记录下来的修改，从而使用户不再需要重新进行代码修改。其中，对生成代码的修改，设想是以命令为单位进行，实时判断命令响应是否有修改，若有，则保存修改的内容，应用修改时，即以保存的修改内容替换自动生成的内容。

参考文献

1 吴同.支持代码自动生成的行为建模.计算机系统应用, 2008,17(6):67－70.
2 廖浩德,杨明根.应用系统自动生成研究.西南民族大学学报(自然科学版), 2007,(4):881－885.
3 曾小宁.数据库应用软件开发中程序代码的自动生成.广西教育学院学报, 2003,(5):37－40.
4 徐爱春,章坚民.基于 XML/XSLT 代码自动生成技术研究.杭州电子工业学院学报, 2004,(4):64－68.
5 陈翔,王学斌,吴全员.代码生成技术在 MDA 中的实现.计算机应用研究, 2006,(1):147－150.
6 杨元峰,赵立杰.基于 XML 的窗体自动生成框架的设计与实现.电脑知识与技术, 2007,19:198－199.
7 Christian Nagel, Bill Evjen. C#高级编程(第 4 版).北京:清华大学出版社, 2006. 593－612.
8 富彧.一个源代码自动生成工具的设计与实现.电脑知识与技术, 2007,(17):1321－1322.
9 袁爱香.基于 MDA 的网上购物系统代码自动生成开发研究.北京联合大学学报(自然科学版), 2008,(1):41－45
10 崔贺超,陈旭东.基于 XML 的网络管理系统界面自动生成技术.计算机时代, 2006,(11):12－14.