测试脚本自动生成器的设计与实现*)

赵斌飞 刘 磊
(吉林大学计算机科学与技术学院 长春 130012)

摘 要 分析了现有的几种测试脚本生成技术,按照 Mosley 的同步数据驱动测试框架(CSDDT)框架设计并实现了一个针对面向对象程序的测试脚本生成器,通过实例验证了方法的可行性和工具的有效性,降低了产生测试脚本的工作量,对已知测试脚本产生过程中的弱点有很好的改进作用。生成脚本可以从单元测试开始应用,重用性良好,可同时测试多个类与方法,无须特定脚本开发语言。

关键词 软件测试自动化,测试脚本 ,面向对象的测试

Design and Implementation of Automated Generator of Test Scripts

ZHAO Bin-fei LIU Lei
(College of Computer Science and Technology of Jilin University, Changchun 130012, China)

Abstract After discussing shortcomings of several available techniques of generating test scripts, the design of a CSDDT-based automated generator and its implementation techniques are introduced. An example illustrates the feasibility of the method and the efficiency of the generator. This technique reduces the workload of testers and improves the present methods. The reusable scripts generated can be applied to several classes and methods at same time from unit testing. There's no need to use new script language during the process.

Keywords Software test automation, Test scripts, OOT

1 引言

测试过程中,在测试人员设计测试用例后,为减少手动执行用例繁重的工作量,通常利用测试脚本来提高测试的自动化程度。

测试脚本在面向对象程序自动化测试过程中起着重要的作用,脚本的设计和创建直接影响测试结果的再现。可见,对测试脚本的自动化创建将极大地提高测试的自动化程度。数据驱动脚本是将数据输入存储在独立的数据文件之中,脚本中只存放控制信息,测试时输入从数据文件中读取,同一个脚本可以运行不同的测试用例,实现了数据与脚本的分离。代表性的数据驱动自动化测试框架是 Archer Group 的同步数据驱动测试(Control Synchronized Data-Driven Testing, CSDDT)[1]。

本文在分析现有的几种测试脚本生成技术之后,基于 CSDDT 框架实现了对于面向对象程序的测试脚本自动生成器,并给出一个应用实例。

2 现有脚本产生技术分析

2.1 线性脚本

当前大部分的测试工具是功能性测试工具,也被称为记录/回放工具。记录/回放工具捕获记录用户的键盘和鼠标动作,以及经过验证的显示输出结果作为回归测试时的基准结果。将人工捕获的信息记录成一个脚本。然后,可以在需要的时候,回放脚本,通过和以前保存的基准结果进行比较,验证当前运行结果。这种脚本通常又被称为"线性脚本"[2]。线性脚本存在很多的应用局限,影响了工具自动功能的发挥,主要表现在以下几个方面:

1)线性脚本的重用性差。对于场景修改不大的程序是有效的,但若对产品进行修改,增强和改变后,捕获的场景不再适用,就需要用户修改捕获的测试脚本,或重新捕获产生新的测试脚本。

2)线性脚本的描述语言基本等同于被测软件的开发语言,要求测试人员对开发工具非常熟悉,才能阅读脚本并对脚本进行有针对性的修改。

3)首次录制时,一般是在对被测软件进行集成测试时,也就是说,被测软件必须已经达到一定的运行稳定性,才能开始录制测试脚本。

2.2 反编译技术

在单元测试中,通常采用反编译技术来产生测试脚本。对于面向对象程序而言,单元通常是指类或其每个成员。使用反编译技术产生单元测试脚本时,首先需要被测单元的源代码,人工选择要测试的单元或方法,然后产生一个或多个单元测试脚本。

通过反编译技术产生的一个测试脚本仅对应一个方法的一次测试,因此,当某个类有许多方法时,该类的单元测试脚本也会很多,导致测试工作量剧增,难以测试所有方法,从而不能突出自动测试的优越性。

2.3 脚本设计语言

采用与测试工具生成脚本的开发语言最接近的语言,借助开发语言手工开发并拷贝部分录制/回放的脚本填充来得到一个优秀脚本,达到脱离记录回放的目的。这要求对测试脚本的开发与被测软件开发同处重要的地位,编程量大,技术要求高,在测试重要性没有被普遍接受的情况下,几乎不可能实现。

3 脚本生成器的设计与实施

由于上述产生技术中,测试数据往往硬编码在产生的测

*)国家自然科学基金青年基金(60603031);博士学科点专项科研基金(20060183044)和吉林省科技发展计划项目(20050527)。赵斌飞 硕士研究生,研究方向为软件测试与质量保证;刘 磊 教授,博士生导师,研究方向为软件形式化、语义网及本体工程。

试脚本中,因此,无论采用上述哪种方式产生测试脚本,一般都需要大量的人工劳动来编辑或调试测试脚本。为保证测试数据的独立,设计出在最新的.NET 框架下对当前脚本产生技术的改善方法,下节将简述相应的实现技术。

3.1 总体设计

总体上生成过程分为以下几个步骤:

1) 对于被测的可执行或动态链接程序集合,获取文件名;

2) 装载程序集合的类型:使用反射(Reflection)机制及 Type 类的方法准确得到被测程序集合的诸多信息,包括被测程序集合中的类型、类型的方法、字段、属性、方法等的参数以及其它相关细节;

3)在电子表格 I 中自动输入获取的类型信息,并为其随机赋初值。这样可避免把测试数据硬编码在测试脚本中,测试数据可以保存在电子数据表中;

4)还可以根据需要从程序集合生成的 XML 注释文档中获取测试用例,动态添加到电子表格 I 中。VS.NET IDE 中,可以用"///"在程序中增加 XML 文档注释,对于开发人员来说,很容易在编码方法时用 XML 注释设定测试用例,而且由开发者在开发时设计的测试用例更有利于揭示错误。编译时,同时产生 XML 辅助文档;

5)自动产生脚本并保存脚本(见 3.2 节);

6)手动运行脚本(见 3.4 节),验证实际结果与预期结果是否相符,生成测试报告。

3.2 脚本产生的自动过程

详述 3.1 节中的第 5 步,具体流程如下:

1)创建一个有序字符流对象用于保存脚本,并制定其文件名;

2)产生需要的各种 CodeDom 对象,用于动态编写脚本内容;

3)为 CodeNamespace 对象增加"对依赖命名空间的引用"的代码语句;

4)为 CodeNamespace 对象增加"测试类(CodeTypeDeclaration 对象)及测试方法(CodeMemberMethod 对象)声明"的代码语句;

5)为 CodeMemberMethod 对象增加"启动电子表格程序,并初始测试报告 II 的标题行"的代码;

6)从保存电子表格 I 中获取类型信息后,为 CodeMemberMethod 对象增加"初始化第一个类型对象"的代码;

7)依次列举所有类型的所有方法的信息(见 3.3 节),再为 CodeMemberMethod 对象添加"启动测试,保存测试结果到测试报告 II 中"的代码;

8)为 CodeMemberMethod 对象添加"保存测试报告 II 文件,并关闭工作表"的代码;

9)把所有已产生的类型(CodeTypeDeclaration 对象)、方法(CodeMemberMethod 对象)等加入到测试脚本中,并为脚本添加一个测试驱动类和一个 Main 方法用于启动测试;

10)关闭存储测试数据的电子表格程序。

3.3 列举所有方法信息

用图 1 中的伪代码算法详述了 3.2 节中的第 7 步执行过程。

```
do
{从电子表格 I 中的第二行依次枚举每行的类型名;
  if(找到新类型)
  then 在脚本中初始化新类型;
  else if(该行方法是构造函数)
       增加测试构造函数的代码;
       else {
       1. 收集该方法参数信息并将方法名字写入脚本中;
       2. 列举参数信息,产生"声明被测方法参数"代码;
       3. try-catch 语句块中产生此方法的测试代码,即用已声明的参数调用该方法;
       4. 测试期望返回值,并将测试结果记录到测试报告 II 中;
       }
}while(!电子表格 I 中所有行被遍历)
```

图 1 列举类型及其方法的算法伪代码

3.4 应用脚本过程

在 VS.NET 中,新建控制台应用程序工程,将产生的测试脚本添加到工程中,再添加电子表格 COM 应用组件引用和待测程序集合引用。编译后,即可自动执行测试任务,该工程与测试数据分离。

如果在命令提示行下键入以下命令:"csc 脚本文件名 /r:本地路径\Microsoft.Office.Interop.Excel.dll /r:本地路径\被测程序集合名",即可得到测试脚本的可执行文件,在命令提示行下运行即可。脚本运行时将读取测试数据,初始化测试报告,并根据类型依次测试每个方法、属性等并保存测试结果到测试报告中。

4 脚本自动生成器的实现技术

.NET 技术 2000 年发布 1.0 版,最新发布 3.0 版。简单地说,它是由一组统一的类库、层次化的服务和开发工具构成,用来克服已有的编程模型的限制。这些服务、类和工具组合在一起而形成了全新的.NET 开发平台。.NET Framework 结构主要包括以下部分:

(1) 公共语言运行时(CLR):公共语言运行时是一个高级运行时引擎,能够提供类型安全、代码校验和异常处理等功能,并且支持多种编程语言。

(2) 基类库(Base Class Library,BCL):提供了一系列功能强大、可扩充的类,这些类适用于所有与.NET 兼容的编程语言。在此基础上,提供了 ASP.NET,ADO.NET,Remoting.NET,Reflection 等功能。

(3) 语言和开发人员工具:.NET 是可以跨语言的,这意味着在.NET 平台支持的语言可以无缝地进行交互。.NET 还提供了全新的编程语言 C#,大幅度简化开发,更适应发展潮流。.NET平台支持语言众多,而且正在不断集成新的语言。

本文在实现脚本自动生成过程中具体使用到了如下的.NET相关技术[3]:

4.1 程序集(Assembly 类)

程序集是.NET 基本的部署单元,它是公用语言运行时(CLR)应用程序的构成组件,配件可以重用。不同的应用可以使用相同的程序集,程序集包含元数据和中间代码和资源,是独立的自描述的基本单元。

4.2 反射(Reflection Namespace)

反射是.NET 提供的非常重要的功能,通过它,.NET 可以访问应用程序的元数据,可以将反射简单地定义为在运行时发现类型信息的能力。生成工具主要用到反射的以下用途:

(1) 使用 Assembly 定义和加载程序集,加载在程序集清单中列出的模块,以及从此程序集中查找类型并创建该类型的实例。

(2) 使用 ConstructorInfo 可得到构造函数的名称、参数、访问修饰符(如 Public 或 Private)和实现详细信息(如 Abstract 或 Virtual)等。

(3) 使用 MethodInfo 了解方法的名称、返回类型、参数、访问修饰符和实现详细信息等。

通过使用 Reflection 技术，不需要被测程序集合源代码，就可以自动获取给定程序集合的类型信息。本文利用反射创建了类型浏览列表，它使用户能够选择要测试类型，然后在电子表格中查看有关选定类型的信息，从而使测试任务能够更快更省地完成。

4.3 代码文档对象(CodeDom)

CodeDom 是实现动态编程的核心。.NET 中包含一个名为“代码文档对象模型”(CodeDOM)的机制，该机制使编写源代码的程序的开发人员可以在运行时，根据代码的模型，用多种编程语言生成源代码。

为表示源代码，CodeDOM 元素相互链接以形成一个“CodeDOM 图”数据结构，它以某种源代码的结构为模型。

4.4 Type 类

Type 是允许多个实现的抽象基类。Type 为 Reflection 功能的根，也是访问元数据的主要方式。使用 Type 的成员获取关于类型声明的信息，如构造函数、方法、字段、属性(Property) 和类的事件，以及在其中部署该类的模块和程序集。例如：Type. GetMethods()返回 MethodInfo 类型数组，该数组收集类型的所有方法；Type. GetFields()方法，返回被测类型的字段信息，保存在 FieldInfo 类型数组中；Type. GetProperties()返回被测类型的属性信息，保存在 PropertyInfo 类型数组中。

5 应用实例

为了说明方法具体应用，定义一个简单的孩子类，如图 2(篇幅原因省略 XML 注释)：Kid 类包含有字段 age(年龄)与 name(姓名)，构造函数，PrintKid(打印孩子信息)和 JudgeKid(判断是否是孩子)两个方法，以及 Details 属性。

```
using System;
namespace KidNSP
{   /// XML 注释
    class Kid
    {   private int age;
        private string name;
        /// XML 注释
        public Kid()
        { name="N/A";
        }
        /// XML 注释
        public Kid(string name,int age)
        {   this. name=name;
            this. age=age;
        }
        // Printing method:
        public void PrintKid()
        { Console. WriteLine("{0},{1} years old. ",name,age);
        }
        /// XML 注释
        public int JudgeKid(string name,int age)
        {   if (age >=20 || age<=0)
                return 0;
            else return 1;
        }
        /// XML 注释
        public string Details
        {   get
            {   return name + "," + age. ToString() + " years old";
            }
        }
    }
}
```

图 2 待测 Kid 类定义

工程自动生成的 XML 注释文档如图 3，其中存放着由编制程序者写入的注释测试数据：7 岁的 Rose 和 45 岁的 Jack。

```
<? xml version="1. 0"?>
-<doc>
-<assembly>
  <name>Kid</name>
 </assembly>
 -<members>
  -<member name="M:KidNSP. Kid. #ctor">
     <summary>Default constructor;</summary>
   </member>
 -<member name="M:KidNSP. Kid. #ctor(System. String,System. Int32)">
    <summary>Instance constructor,Initiate a Kid object</summary>
    <param name="name">assign a string,e. g. ,Rose</param>
    <param name="age">assign an integer,e. g. ,7</param>
  </member>
 -<member name="M:KidNSP. Kid. JudgeKid(System. String,System. Int32)">
    <summary>Conduct a simple judgement operation</summary>
    <param name="name">assign a string,e. g. ,Jack</param>
    <param name="age">assign an integer,e. g. ,45</param>
   <returns>Returns a 0 or 1 result,e. g. ,0</returns>
  </member>
 -<member name="P:KidNSP. Kid. Details">
    <summary>Get accessor,Read only property</summary>
   </member>
  </members>
</doc>
```

图 3 源码中的注释，包含测试数据

利用开发的自动工具将包含在 XML 文件中的测试数据自动填入到电子表格中，产生文件如图 4：7 岁的 Rose 是构造函数的测试数据，45 岁的 Jack 是 JudgeKid 方法的测试数据。

	A	B	C	D	E
1					
2	Kid	Kid			
3	Kid	Kid	Rose	7	
4	Kid	PrintKid			
5	Kid	JudgeKid	Jack	45	
6	Kid	get_Details			
7	Kid	GetType			
8	Kid	ToString			
9	Kid	Equals	object obj		
10	Kid	GetHashCode			

图 4 从 XML 中读出的测试数据

为说明错误提示方式，故意在 JudgeKid 方法的代码中引入逻辑错误，使年龄大等于 20 岁时，返回 1(是孩子)，而小于 20 岁时返回 0(非孩子)。而测试数据中对 45 岁的 Jack 判断后的期望返回值应该为 0(非孩子)。带有错误提示的测试报告如图 5。

	A	B	C	D	E	F	G
1	被测方法	结果	错误信息	实际返回值	期望返回值	参数及其值	
2	Kid	通过	没有	Test Constructor			
3	Kid	通过	没有	Test Constructor			
4	PrintKid	通过	没有				
5	JudgeKid	通过	没有	1	0	name = Jack	age = 45
6	get_Details	通过	没有	Rose, 7 years old			
7	GetType	通过	没有	KidNSP. Kid			
8	ToString	通过	没有	KidNSP. Kid			
9	Equals	通过	没有	FALSE		obj = object obj	
10	GetHashCode	通过	没有	45653674			

图 5 实际值与返回值冲突报告

再将 JudgeKid 代码改回正确的逻辑，年龄大于 0 小于 20 时为孩子。产生预期返回值与实际返回值相符的测试报告如图 6。

	A	B	C	D	E	F	G
1	被测方法	结果	错误信息	实际返回值	预期返回值	参数及其值	
2	Kid	通过	没有	Test Constructor			
3	Kid	通过	没有	Test Constructor			
4	PrintKid	通过	没有				
5	JudgeKid	通过	没有		0	name = Jack	age = 45
6	get_Details	通过	没有	Rose, 7 years old			
7	GetType	通过	没有	KidNSP.Kid			
8	ToString	通过	没有	KidNSP.Kid			
9	Equals	通过	没有	FALSE		obj = object obj	
10	GetHashCode	通过	没有	45653674			

图 6　无错的测试报告

再将 JudgeKid 中加入以下引起错误的语句：

String a=null;

Console. Write(a. Length);

使其创建对象过程失败，得到方法调用失败的测试报告如图 7。

	A	B	C	D	E	F	G
1	被测方法	结果	错误信息	实际返回值	预期返回值	参数及其值	
2	Kid	通过	没有	Test Constructor			
3	Kid	通过	没有	Test Constructor			
4	PrintKid	通过	没有				
5	JudgeKid	失败	未将对象引用设置到对象的实例。		0	name = Jack	age = 45
6	get_Details	通过	没有	Rose, 7 years old			
7	GetType	通过	没有	KidNSP.Kid			
8	ToString	通过	没有	KidNSP.Kid			
9	Equals	通过	没有	FALSE		obj = object obj	
10	GetHashCode	通过	没有	54267293			

图 7　方法调用失败的报告

结束语　为减少生成脚本过程的工作量，基于同步数据驱动测试框架，设计并实现了面向对象程序的测试脚本自动生成工具。生成的脚本直接从底层测试开始应用，重用性良好，可同时测试多个类与方法，无需特定脚本开发语言。对已知测试脚本产生过程中的弱点有很好的改进作用。当然利用. NET 技术还能够将更多的测试活动自动化。本文关注的是对面向对象的程序集自动产生测试脚本的改善方法。今后主要工作是对复杂结构类型的处理和 GUI 交互程序的测试脚本自动产生策略。

参 考 文 献

[1] Mosley D J, Posey B A. 软件测试自动化[M]. 邓波，黄丽娟，曹青春，等译. 北京：机械工业出版社，2003

[2] Fewster M, Graham D. 软件测试自动化技术与实例详解[M]. 舒智勇，包晓露，焦跃，等译. 北京：电子工业出版社，1999

[3] Schmidt M, Robinson S. Microsoft Visual C# . NET 2003 Developer's Cookbook [M]. Sams publishing, 2004

[4] Juristo N, Moreno A M, Strigel W. Software Testing Practices in Industry [J]. IEEE Software, July 2006: 19-21

[5] Kanglin L, Mengqi W. Effective Software Test Automation [M]. Alameda, CA: Sybex, 2004

[6] MSDN Library for Visual Studio [EB/OL], 2007

[7] Perry W E. Effective methods for software testing [M]. New York : Jonn Wiley & Sons, 2000

(上接第 257 页)

根据人脸梯度特点，设定适当的梯度阈值后，在眉、眼、鼻、嘴所在的部位应存在较大梯度(白色)，而脸颊和额头部位梯度很小。由此可建立一个简单的人脸梯度模板，如图 9 所示。在实际应用时，应允许人脸上五官的位置在一定范围内存在偏差，以用于检测人脸有转动和倾斜的情况。

图 11　视频中检测出的人脸

结合水平亮度投影，根据梯度模板中各五官的分布特点，即可在人脸候选区域中确认人脸。采用水平亮度投影和特征点的梯度模板对图 5 进行了运算，其结果如图 10 所示。视频中检测出的人脸如图 11 所示。

结束语　本文对复杂背景下视频序列中的人脸定位方法进行了研究与描述，利用视频序列中人脸的特点，提出了对应于视频序列中的人脸定位算法。通过实验验证，该方法用奔Ⅳ2.8G 的计算机利用 C# 编程，对复杂背景下以正常速度基本保持正面通过视频摄像区域的人脸进行定位，单张脸检测耗时约 60ms，能够顺利检测出通过摄像区的人脸，误检率小于 5%，完全能够满足人脸门禁系统的要求。目前，我们已利用本文研究的成果开发了数字智能多功能门禁与报警系统。

参 考 文 献

[1] Yang M H. Recent advances in face detection [A] // Proc. of the IEEE ICPR 2004 Tutorial [C]. United Kingdom (Cambridge): IEEE, 2004

[2] Yang M-H, Kriegman D J, Ahuja N. Detecting faces in images: A survey [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24(1): 34-58

[3] Leung T K, Burl M C, Perona P. Finding faces in cluttered scenes using random labeled graphic matching [C] // Fifth International Conference on Computer Vision, Cambridge . M. A. June 1995

[4] Feraud R, Bernier O J, Viallet J E, et al. A fast and accurate face detection based on neural network [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2001, 23(1): 42-53

[5] Hsu R-L, Abdel-Mottaleb M, Jain A K. Face detection in color images [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24: 696-706

[6] Hsu R L. Abdel-Mottaleb M, Jain A K. Face detection in color image. http://www.image2003.com