TTCN-3 语言非侵入式调试技术①

赵 营，程绍银，蒋 凡

(中国科学技术大学 计算机科学与技术学院, 合肥 230027)

摘 要：TTCN-3(Testing and Test Control Notation version 3)是一种形式化的测试描述语言，被广泛应用于协议测试等领域。由于测试套的复杂度越来越高，测试人员迫切需要能够调试 TTCN-3 语言的技术。本文提出了一种非侵入式的 TTCN-3 语言调试方法，通过驱动第三方调试器对 TTCN-3 语言编译后生成的目标代码进行调试，同时结合语句映射和符号解析等算法，实现了基本的调试功能。实验结果表明，该调试技术对测试系统的执行性能影响非常小，能更好的满足对大型测试套的调试需求。

关键词：TTCN-3；编译；调试；测试

Non-Intrusive Debugging Methods of TTCN-3

ZHAO Ying, CHENG Shao-Yin, JIANG Fan

(School of Computer Science & Technology, University of Science & Technology of China, Hefei 230027, China)

Abstract：TTCN-3 (Testing and Test Control Notation Version 3) is a formalized testing description language, which is widely used in protocol testing and other testing fields. As TTCN-3 test suites are more and more complex, it becomes difficult for testers to locate the place which causes testing failures without debugger's help. This paper presents a non-intrusive TTCN-3 debugging method which uses third-party debugger to debug TTCN-3 complier generated target code. In addition, the method integrates statement mapping and symbol analysis algorithms to implement basic debugging functionality. Experiments show that this method imposes little influence on testing system and has a better performance while debugging large-scale test suite.

Key words：TTCN-3; compile; debugging; testing

TTCN-3(Testing and Test Control Notation Version 3)语言是由欧洲电信标准协会 ETSI 推出的国际标准测试语言。与其他通用语言不同，该语言具有测试专用的语言元素[1]，具有强大的测试过程描述能力和广泛的通用性，因而被大量应用于协议测试等领域。

现今协议测试在逐渐向大型化、复杂化方向发展，寻找测试失败的原因也变得越来越困难。一般而言，定位问题的最直接方法是通过调试来追踪测试脚本的实际执行流程，在运行过程中发现测试异常之处。然而调试是 TTCN-3 工具中比较难以实现的功能之一，国内在相关领域内的研究比较少。文献[2]提出了一种调试器设计方案，采用的是一种侵入式调试方法，会在较大程度上增加测试套的执行时间。国外有比较成熟的 TTCN-3 工具像 TTWorkBench 和 IBM Tau 都实现了常用的调试功能，但二者是商业软件，价格昂贵且技术不公开。

本文对 TTCN-3 调试问题进行了深入分析，针对 TTCN-3 语言的编译执行方式提出了一种非侵入式的 TTCN-3 语言调试技术，文中对技术架构进行了详细介绍，并阐述了主要的实现算法。最后的实验表明，该技术对测试套执行性能的影响甚小，能够取得比较好的调试效果。

① 基金项目:中央高校基本科研业务费专项资金(WK0110000007)

收稿时间:2011-09-20;收到修改稿时间:2011-10-23

1　问题分析

TTCN-3 是一种形式化的测试描述语言，由它所编写的测试代码称为抽象测试套(ATS, Abstract Test Suite)。ATS 可以解释执行，也可以编译执行。编译执行通常都是将其翻译成另外一种高级语言如 C++[3]、Java，再使用生成的文件与运行期支持库一齐构成可执行测试套 TE[1] (TTCN-3 Executable)，由测试系统加载执行。编译执行的过程如图 1 所示，其中测试系统中的 TM(Testing Management)负责测试管理，如启动测试例等；CH(Component Handler)负责组件管理；CD(Coder & Decoder)负责按协议规定的消息格式对消息进行编解码；SA(System under test Adapter)负责与被测系统的通信；PA(Platform Adapter)负责实现平台特有的时钟功能及一些平台相关的函数。

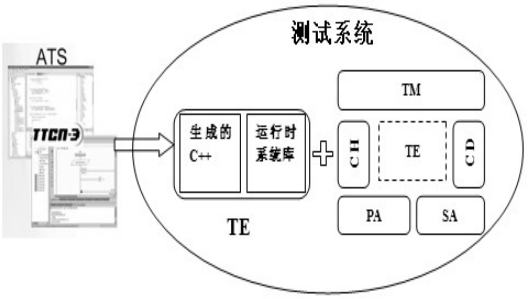

图 1　ATS 编译执行过程

由上可知，TTCN-3 代码执行方式比较特殊，故其调试方法也异于其他编程语言。文献[2]提出了一种 TTCN-3 调试器设计方案：在翻译产生的 C++代码中写入调试信息，使调试器作为 TE 的一部分，以便于实现对 TTCN-3 的执行控制。但运行时调试一般要遵循非侵入原则[4]，即调试程序不能过度地影响被调试程序的运行，更不能改变程序原有的行为。[2]中的方法在 TE 中增添额外的代码，既增加了执行时间，又有可能造成在调试状态下与非调试状态下运行效果不一致，属于典型的侵入式调试。因而该方法功能受限，尤其不适用于大型测试套和对实时性有要求的测试。

本文提出了一种非侵入式的调试技术，其实现方式如下：用户发出调试命令后，调试系统通过符号解析将命令中的 TTCN-3 符号转化为 C++符号，然后驱动第三方 C++调试器完成相应的功能；C++调试器做出响应后，调试系统再将反馈结果中的信息映射回 TTCN-3 符号展现给用户。该技术的整体框架如图 2 所示（图中箭头表示数据流向）。

在该框架图中，最上面的是用户接口部分，负责接收用户命令、展示调试结果；中间部分是调试系统的核心，负责处理运行时的调试信息；最下面的是本地 C++调试器，用于调试翻译生成的 C++代码。

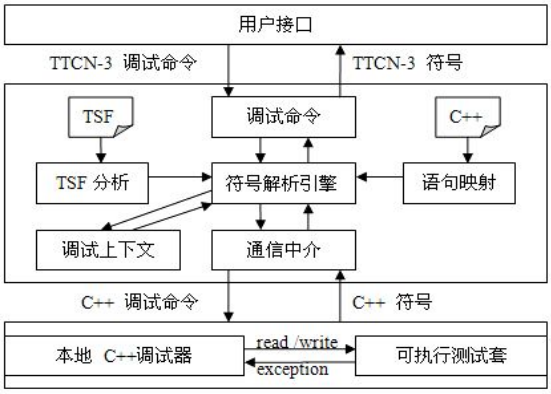

图 2　技术框架图

调试系统核心主要由调试命令、符号解析引擎、调试上下文、通信中介及其他模块构成。调试命令模块负责调试命令的具体执行过程；符号解析引擎负责向其他模块提供统一的符号解析和语句映射接口，由语句映射器和 TSF (TTCN-3 Symbol File)文件解析器辅助实现相应功能。TSF 文件包含的是编译期间 TTCN-3 编译器的符号表的内容。调试上下文环境用于记录当前调试过程的运行状态信息，包括行号、文件、断点位置、断点集合、被调试程序的状态、行为体、调用堆栈、主模块等。通信中介通过管道与本地 C++调试器进行通信，传送命令给 C++调试器并驱动 C++调试器对可执行测试套进行调试。

这种方法的优点在于：1) 调试系统独立于 TE，不需要向 TE 中加入调试信息，属于非侵入式调试。2) 使测试系统的翻译机制对用户透明，用户不需要理解底层的 C++代码，只需关注 TTCN-3 代码的执行过程。

2　技术实现

本文所提出的调试技术的实现难点主要在于 1)由于受翻译方案所限，TTCN-3 代码与翻译生成的 C++代码行之间不存在规则的映射关系，需要设计语句映射算法，来找到 TTCN-3 语句与 C++语句之间的准确映射关系。2)TTCN-3 和 C++使用的是不同的符号信息，需要通过符号解析来进行符号的转换。

2.1 语句映射

语句映射即 TTCN-3 语句与 C++语句之间的对应

关系。由于一条 TTCN-3 语句可能被翻译成为多条 C++语句，因而一条 TTCN-3 语句可能对应 N(N>=1)条连续的 C++语句；但不会存在两条及以上 TTCN-3 语句对应同一条 C++语句的情况。

在进行语句映射之前，首先要对语句的位置进行定义。一条 TTCN-3 语句的位置可由一个三元组确定，即<文件名.行号.偏移量>，可表示为<file.line.offset>。其中偏移量是指该语句是所在行中的第几个，因为同一行中允许存在多条语句。C++语句的位置可由两部分确定即<文件名.行号>，这是因为翻译方案是可控的，能够确保在同一行中只产生一条 C++语句。

把 TTCN-3 语言中的元素分为两大类：第一类没有实际的执行语义，无法对这些元素进行执行时控制，记为类 ω0={类型定义，组件定义，常量定义、模板定义、模板参数、测试例定义、函数定义、可选步定义}。第二类是具有执行语义的元素集合，记为类 ω1={基本程序语句、行为语句、配置操作、通信操作}。

设 TTCN-3 测试套 T 由 n 个 TTCN-3 脚本构成，表示为 $T=\{t_1,t_2,\cdots,t_n\}$，其中 t_i 为脚本文件名。语句映射算法如下：

输入：TTCN-3 文件集合 $T=\{t_1,t_2,\cdots,t_n\}$

输出：1) C++文件集合 $C=\{c_1,c_2,\cdots,c_n\}$，c_i 是由脚本 ti 翻译生成的 C++文件。

2)对 t_i 中的任意语句<t_i.line.offset>$\in \omega_1$，给出其在翻译文件中的起始位置<c_i.line’>。

```
BGEIN
1)
for each ti in T
{
对 ti 进行词法、语法分析和静态语义检查，并查看当前元素结点的属性
if (当前元素<ti.line.offset>∈ω1)
{
在翻译文件中增加标记语句 mark(line.offset)
//表示此句以下 C++语句系由 line.offset 翻译而成
正常翻译
 }
2)初始化向量 vec
for each ci in C
{
    初始化映射 mi
    对 ci 的每一行进行正则表达式匹配，寻找标记点 mark(line.offset)
        if(找到标记语句)
        {
         获取标记语句所在的 C++行号为 line’
         mi.insert(line.offset，line’);
         }
    vec.push_back(mi);
    }
    END
```

在上述算法中，每个 TTCN-3 文件中的可执行语句的映射关系构成了一个行域，这些行域被保存到一个向量之中，以便于快速查询。

2.2 符号解析

TTCN-3 语言中的符号的信息包括类型(变量、函数、类型、模板等)、作用域、位置等。符号解析包括两个方面，一是将 TTCN-3 中的符号转化为执行时 C++中的名称、类型及该类型对应的数据操作接口；另一方面是逆向解析，即将 C++中的符号转化为 TTCN-3 中的对应体。由于符号解析方法依赖于所采用的翻译方案，因而是与具体的 TTCN-3 实现工具密切相关的。本文以中科大的 TTCN-3 测试平台 LoongTesting[3][5]为例，说明符号解析的过程。LoongTesting 在翻译过程中定义了若干条转换规则，其中主要有：

Rule 1.名字修饰策略：翻译方案中对 TTCN-3 中的符号名称进行重命名的规则。

Rule 2.类型对应规则：TTCN-3 中的类型与可执行测试套中的真实类型的对应关系。

Rule 3.数据操作接口对应规则：操作 TTCN-3 中某种类型的值所对应的一系列 C++接口的集合。

下面以 TTCN-3 符号到 C++的转换算法为例，说明符号解析的过程。

输入：TTCN-3 标识符：identifier

输出：C++中对应的符号信息

BEGIN

1) 编译期间将符号表信息导出到自定义格式的 TSF 文件中。

2) 将 Rule 1—Rule 3 中的规则形式化之后存储到符号解析引擎之中。

3) 根据当前运行时作用域，在 TSF 文件中寻找名为 identifier 的符号。

```
if (not found)
    转到出错处理
```

else

符号解析引擎依次对 identifier 应用 Rule 1、Rule 2、Rule3 获取并返回 C++对应的名称、类型、及操作接口

END

从 C++到 TTCN-3 符号的转换算法与此类似，这里不再赘述。解决语句映射和符号解析问题之后，就可以对 TTCN-3 进行程序控制并利用符号解析的结果进行数据操作。图 3 是设置 TTCN-3 断点的执行流程。

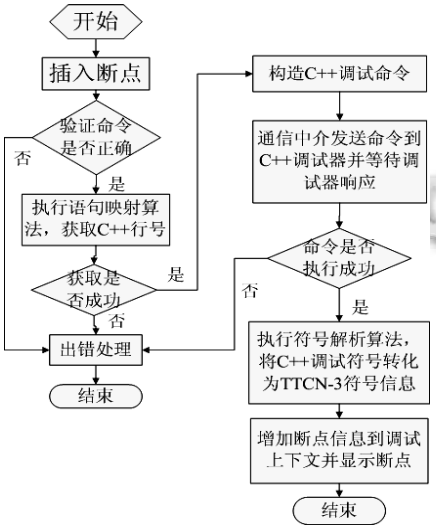

图 3　设置断点流程

3　实验

本实验主要关注调试对原有测试套执行性能的影响。实验使用了三个 TTCN-3 测试套，分别是 Echo、BookStore 和 SIP[6]测试套，它们的复杂度依次增加。其中 Echo(约 110 行)是回音测试，只简单地进行消息收发；BookStore(约 2,000 行)用于测试一个网上书店，属于 Web 测试; SIP 测试套(约 60,000 行)用于测试 SIP 协议，测试过程比较复杂，测试脚本可以在[7]获得。最终的实验数据分析结果如图 4 所示，图中方法 1 是[2]中提出的方法，方法 2 是本文提出的方法。图中纵坐标是：执行时间增加比例=(调试模式下执行时间-正常模式下执行时间)/正常模式下执行时间*100%。

由图 4 可以看出，方法 1 产生的时间增加比例会随着测试套规模的变大而线性增加，这是因为随着测试套的规模的扩大，写入到 TE 中的额外信息会越来越多。而方法 2 当测试套规模的变大时调试产生的时间增加比例会减少，这是因为该方法使调试系统独立于 TE，由调试引入的时间开销基本是固定的，故当测试套规模变大时该时间开销所占比重会变小。

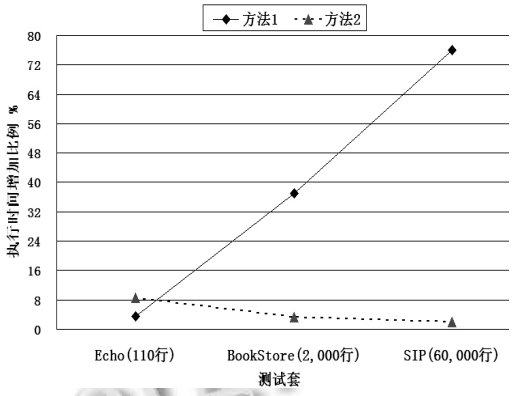

图 4　实验结果

从本实验可以看出，方法 2 产生的执行时间增加比例要小得多，在调试大型测试套时尤为明显：对 SIP 测试执行时间增加了 3%左右，几乎可以忽略。

4　总结

如何有效调试 TTCN-3 代码一直是测试人员关注的问题，针对现有调试方法的不足，本文提出了一种非侵入式 TTCN-3 语言调试技术。该技术可大大减少对测试系统执行性能的影响，同时使测试系统的翻译机制对用户保持透明。所设计的解决方案的架构易于理解，其中采用的语句映射和符号解析算法具有一定的通用性，对其他依靠翻译执行的语言的调试也有一定的借鉴意义。

参考文献

1 ETSI ES 201 873-1 V4.1.1. Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; PArt 1: TTCN-3 Core Language. 2009-06.

2 王建学.TTCN-3 语言调试器设计与实现[硕士学位论文].合肥:中国科学技术大学, 2009.

3 张辉，蒋凡.基于 C＋＋语言转换的 TTCN-3 测试系统的设计与实现.计算机系统应用,2007,16(10):64–67.

4 Jonathan B, Rosenberg. How debuggers work: Algorithms, data structures and Architecture. New York: Wi1ley Computing Publishing,1996:7–11.

5 http://ttcn.ustc.edu.cn/.

6 RFC3261, http://www.ietf.org/rfc/rfc3261.txt, Accessed 8 April 2011RFC3261.

7 http://www.ttcn-3.org/.