对 AspectJ 获取逆向工程所需基本信息的研究①

马 骕[1]，李青山[2]，陈鹏岗[3]

[1](西安工程大学 计算机学院，西安 710048)
[2](西安电子科技大学 软件学院，西安 710071)
[3](西安交通大学 医学院第二附属医院 信息网络部，西安 710004)

摘 要：通过利用 AspectJ 获取逆向工程所需的信息，实现基本信息的提取。作为 AOP 具体实现之一的 AspectJ 对 Java 程序进行分析，将所需的各种信息提取出，并保存到文本文件提供给下一个功能模块进行格式转化。要提取的基本信息包括调用、返回和返回值三个方面。

关键词：逆向工程；AOP；AspectJ；横切；横切关注点

Basic Information Needed in Reversing-Engineer by AspectJ

MA Su, LI Qing-Shan, CHEN Peng-Gang

[1](College of Computer Science, Xi'an Polytechnic University, Xi'an 710048, China)
[2](College of Software, Xidian University, Xi'an 710071, China)
[3](Second Affiliated Hospital Medical College, Xi'an Jiaotong University, Xi'an 710004, China)

Abstract: AspectJ, which is one of concrete realization of AOP, can analyzes Java-procedure. We abstract various necessary information, then save it to the textfile and offer the next format-transfering function module. The information to be abstracted includes three aspects: calls, returns and returned values.

Keywords: reverse engineering; AOP; AspectJ; crosscut; crosscut attention

随着计算机科学技术的进一步发展，无论在数量上还是技术上都对软件开发提出了更高的要求。面向对象的编程方式 OOP(Object－Oriented Programming)技术的出现能更好地处理一般行为，但是，OOP 不能很好地处理跨越多个、经常是不相关的模块行为。此时， AOP(Aspect－Oriented Programming)面向方面的编程方法应运而生，它能够利用模块化来分离软件中横切多模块的关注点。AOP 的开发环境——AspectJ 工具，它是 Java 语言的扩展，提供了一整套的语法，能够清楚地描述横切关注点，并将其植入到 Java 源代码中。本文就是利用 AspectJ 对 Java 程序进行分析实现基本信息的提取，要提取的基本信息包括调用、返回和返回值三个方面，这些都是程序在运行过程中产生的动态信息，记录的是本次运行时出现的各种关系。

1 Java和AspectJ

AspectJ 是 Java 的 AOP 实现，是一个多功能的面向方面的 Java 扩展。它使用 Java 作为单个关注点的实现语言，并扩展了 Java 来指定植入规则。这些规则是用横切关注点(Pointcut)、连接点(Join Point)、通知(Advice)和方面(Aspect)来说明的。连接点是定义在程序执行过程之间的点，横切关注点是由指定连接点的语言构造，通知定义了在横切关注点上执行的代码片，而方面则是这些基础元素的组合。AspectJ 的植入器和编译器负责把不同的方面组合在一起。由 AspectJ 编译器建立的最终系统是纯 Java 字节码，可以运行在任何符合 Java 标准的虚拟机上。

2 AOP和AspectJ

AOP 提供了一种描述横切关注点(Pointcut)的机

① 基金项目:国家科技支撑计划(2006BAF01A44,2007-2009);陕西省教育厅项目(2010JK562)
收稿时间:2010-05-31;收到修改稿时间:2010-07-03

制，并能够自动将横切关注点植入到面向对象的软件系统中，从而实现了横切关注点的模块化。通过划分 Aspect 代码，横切关注点变的容易处理。开发者可以在编译时更改、插入或除去系统的 Aspect，甚至重用系统的 Aspect。使用 AOP 可以建立容易设计，易于理解和维护的系统。

AOP 有两个基本的术语：Pointcut 和 Advice。可以用事件机制的 Event 和 Action 来类比理解。Pointcut 类似触发器，是事件 Event 发生源，一旦 Pointcut 被触发，将会产生相应的动作 Action，这部分 Action 称为 Advice。Advice 在 AspectJ 中有三种：Before、 After 和 Around，它是真正的执行代码，或者说关注的实现，类似 Action。Join Point 是代码中激活 Advice 被执行的触发点，一系列的 Join Point 称为 Pointcut。因此，AspectJ 可实现切面(Aspect)式的编程。

3 AspectJ对逆向工程的支持

逆向工程是从最终产品推断出设计方案，目的是为了恢复错误的，不完整的或难以获得的文件，正被研究应用于老的软件系统。逆向工程并不改变目标系统，是一个检查的过程，而不是修改的过程。逆向工程通过标识对象、发现其间关系并抽象系统，从而辅助对系统的理解。例如：数据收集是逆向工程的一项基本活动，它采用的技术包括对程序源代码进行静态分析、动态分析和获取准确和可靠的数据等，因为原始数据就是构造和浏览高层抽象的基础。

AspectJ 本身并不能进行逆向工程，但它可以通过定义的方面 Aspect 对源程序进行反射，并且利用 ajc 编译器来编译类和 Aspect 代码。生成有效的.class 或.java 文件，也可以作为预编译器操作，获取程序的静态信息、动态信息。通过 Aspect 将植入的软件触发器反射到源程序中，这些行为都可以在编译时进行，不会增加程序的运行开销。

4 利用AspectJ进行静态分析与动态分析

静态解析：静态解析机制的职能主要是为动态信息过滤提供依据，以及实现程序多维分解和分层抽象。通过对某种特定语言的源码或编译后的目标码进行静态解析，将所需的类型信息、函数接口、函数调用关系、程序中的数据流和控制流等信息，从静态解析结果中分离出来，再转换成相应的静态模型表示。

软件触发器：是在源程序中相应的位置添加的一些代码，运行时由这些代码按特定协议将指定的动态信息传递到指定位置或传递给动态信息收集机制，从而提供产生动态模型所需的对象之间的消息传递。

交互子系统与植入子系统之间的接口：提供了植入位置和植入内容，是通过植入位置信息文件和植入内容信息文件来表示的。如果将文件内容用元数据来描述，当植入内容和植入位置发生变化时，不需要直接修改元程序，只需修改元数据文件，只有在发生很大变化时，才去修改元程序。这样可以减少代码长度，提高程序的灵活性，方便程序的维护和扩充。

静态解析与植入的设计

AspectJ 的 Aspect 为实现逆向工程提供了支持，AspectJ 是典型的开放式编译器，利用 AspectJ 的 ajc 可以访问 Aspect、Java 程序的编译信息，获得对程序本身的描述，将类之间的关系、函数接口、参数、函数所在的类等信息提取出来。

用 Aspect 来实现软件触发器的植入机制，是通过 Aspect 作用于编译器。这样，应用程序的植入过程是在编译时进行的，在交付编译之后，由 Aspect 对应用程序植入软件触发器，再透明地向常规编译提交，链接时链入所需的运行时支持机制(如动态信息协议的实现等)，这就达到了将植入的软件触发器和被植入的应用程序作为两个计算层次来处理的目的。

静态解析与动态植入的类图如图 1 所示。

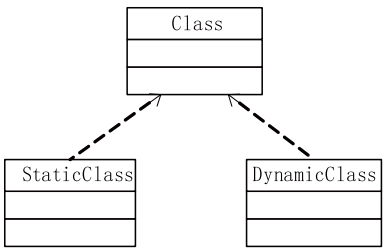

图 1 静态解析与植入的类图

其中，Class 是 AspectJ 提供的默认的元类，StaticClass 类是进行静态解析的元类，DynamicClass 类是进行动态植入的元类，它们继承 Class 类，实现对源程序的静态解析与植入软件触发器。

静态解析与植入的实现

① 静态解析的具体实现

对 Java 源程序进行静态解析，该源程序就称为基级程序。首先要为需要进行解析的类指明元类 StaticClass，元类所在的程序称为元程序，所有与获取静态信息相关的操作都在元类中定义，所有的元类都必须继承 AspectJ 缺省的元类 Class，元类 StaticClass 重置 Class 中的成员函数，获取所需的信息。

元类的声明方式：

profeclass classl class2；

classl 是指明的元类，class2 是被反射的类。

在定义的 AspectJ 方面中，利用 TranslateClass()和 TranslateMemberFunction()来获取静态信息。获取类的静态信息的主要步骤描述为：1. 在元类 StaticClass 中重置函数 TranslateClass()的定义；2. 利用类的元对象上的成员函数 Name()获得类名；3. 调用类的元对象的成员函数 BaseClasses()获得父类名；4. 调用 ToString() 转换为 char，转换后的两个字符串连在一起，中间以分隔符“#”分开；5. 结果写在静态信息文件中。

文件中对类的描述形式为：

CLASS # 类名 # 继承方式 父类名。

以上步骤如图 2 所示。

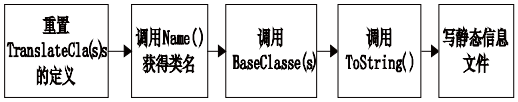

图 2 获取类的静态信息

获取函数调用信息的主要步骤描述为：1). 元类 StaticClass 中重置 TranslateMemberFunction()的定义；2). 获取函数所在的进程号；3). 获取关键字；4). 获取主调函数名 Name()及参数类型和参数；5). 调用被调函数所在的类，获取类名 Ename()；6). 调用被调函数所在的对象，获取对象名 Eobject()；7). 获取被调函数名 Efunction()及参数类型和参数；8). 以上得到的结果写入动态信息文件。

文件中对函数的描述形式为：

进程号 # 关键字 # 主调函数名(参数类型 参数 …)# 被调函数所在的类 # 被调函数所在的对象 # 被调函数(参数类型参数…)

以上步骤如图 3 所示。

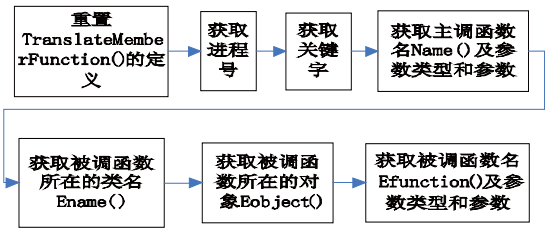

图 3 获取函数调用的动态信息

② 植入软件触发器的实现

AspectJ 中类的元对象可以改变基级程序的行为，如在本文中定义的元类 DynamicClass，实现触发器的植入。元类的设计属于程序设计，可以由用户进行修改，可以根据需要定义几种元类，来控制对程序的植入。植入的内容在元类中定义，而且元类具有通用性，可以针对多个类，不需要对每个类都定义元类。经过 AspectJ 编译后，植入的内容就反映在应用程序中。

代码植入主要与 TranslateClass() 和 TranslateMemberFunction()这两个接口相关，下面对关键问题的思想进行描述(通过例子来说明)，例如：

```
public class Point
{int x;
int y;
Point() { x＝0； y＝0；};
public void Move (int newx, int newy)
{ x = newx;
y = newy;
}
}
```

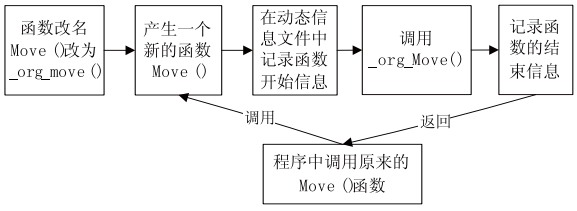

图 4 植入原理

将成员函数 Move()改名为_org_Move()，并为_org_Move()产生一个新的函数 Move()。这样，对原来的函数 Move()的调用，被新的 Move()函数截获，然后由 Move()调用_org_Move()，在 Move()中实现对_org_Move()函数信息的记录。例如，函数的开始、结

束、返回值，这些记录保存在动态信息文件中，植入原理如图 4 所示。

成员函数的改名由 TranslateClass() 和 TranslateMemberFunction() 实现，新函数的生成由 TranslateClass()实现，植入的具体步骤描述为：1). 在元类 DynamicClass 中重置 TranslateClass()；2). 判断类的每一个成员是否为函数，若是，定义一个成员对象，来构造新函数；3). 为旧的成员函数改名；4). 将改变反映在类的定义中；5). 构造新函数的函数体；6). 将新函数的定义反映在类中。

以上步骤如图 5 所示。

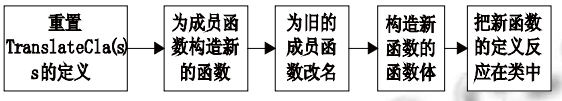

图 5　植入步骤

植入代码并不是直接植入到源文件中，而是生成一个中间文件，后缀名仍为 java 源文件的内容以及要植入的代码写在其中，再由编译器编译这个中间文件，生成可执行程序。程序运行后，会产生动态信息文件。

5　信息获取示例

测试代码

testmain.java

```
package com.ms.apj.apjtest;
import com.ms.apj.data.*;

public class testmain{
public testmain(){
}
public static void main(String[] args){
try{ test t;
t=new test(1);
t=new test();
try {
t.write();
}catch(Exception e2){System.out.println("catch");}
Data dt=new Data();
dt.x=56;
dt.sdata="hello";
String s=dt.sdata;
}
catch(Exception e){System.out.println(e);}
System.out.println("process end\n");
}
}

class test{
test(){
}
test(int i){
// System.out.println("new test");
}
void write() throws Exception{
System.out.println("write to con");
throw new Exception("hhh");
}
}
```

该测试代码在 com.ms.apj.apjtest 包中，导入 com.ms.apj.data.*，所以在运行时，data.java 和 testmain.java 必须放在同一目录下，该测试例中包含有异常、方法的执行、方法的调用及构造子(对 test 的重置)等。看上去简单明了，但可以充分说明植入问题，是一段较好的测试代码。

输出结果：

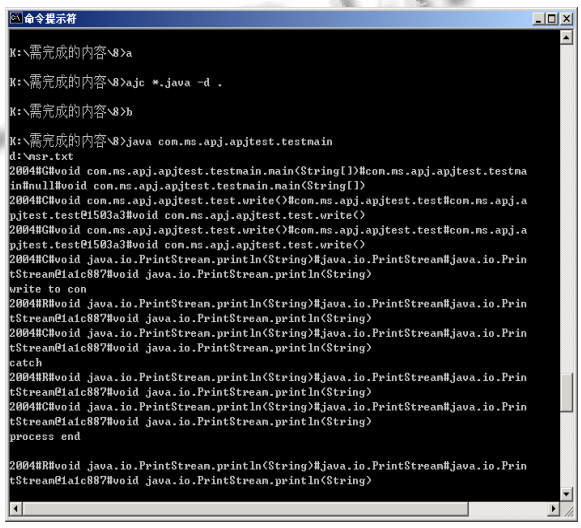

图 6　基本信息输出结果

分析输出结果：

例 1：

```
2004#G#*#com.ms.apj.apjtest.testmain#Global#
  void om.ms.apj.apjtest.testmain.(Smtarining)[]
```

图 7 全局函数信息

代码行中：2004 是进程号，“#”为分隔符，关键字 G 是全局函数，“*”表示缺省状态，com.ms.apj.apjtest.testmain 是入口函数所在的类，Global 是入口函数所在的对象，void com.ms.apj.apjtest.testmain.main(String[])是入口函数。该语句在输出结果中不具有随机性，是特定为下一格式转化模块提供的一条固定输出语句。

例 2：

```
2004#C# void java.io.PrintStream.println(String)#
java.io.PrintStream#java.io.PrintStream@1a1c887#
void java.io.PrintStream.println(String)
write to con
2004#R# void java.io.PrintStream.println(String)#
java.io.PrintStream#java.io.PrintStream@1a1c887#
void java.io.PrintStream.println(String)
```

图 8 调用、返回及返回值信息

上述代码行中：2004 是进程号，“#”为分隔符，C 关键字是调用，R 关键字是返回，void java.io.PrintStream.println(String) 是调用函数名，java.io.PrintStream 是被调函数所在的类，java.io.PrintStream@1a1c887 是被调函数所在的对象，void java.io.PrintStream.println(String)是被调函数。

测试界面

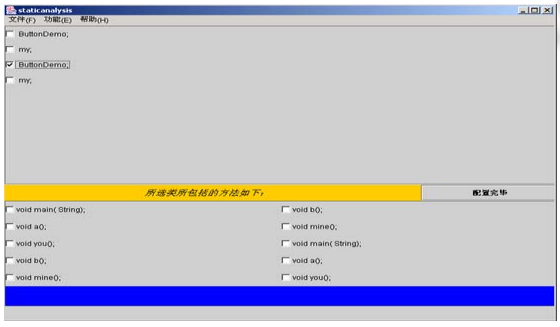

图 9 测试界面

当进行信息提取后，将捕获到的动态信息文本文件通过定义的接口(本文以 com.ms.apj 包来实现)，提供给实现界面模块。通过测试在界面中能显示其结果，并检验其正确性。

实验及测试结果表明：利用 AspectJ 对 Java 程序进行分析之后，可以收集到程序当中的基本信息，为调用、返回及返回值三个方面。这些信息都是程序在运行过程中产生的动态信息，具体包括进程号、关键字、主调函数名、被调函数所在的类、被调函数所在的对象和被调函数，以及在深入研究后得到的方法执行和构造子信息。

6 展望

关于 AspectJ 的开发研究目前还处于初级探索阶段，有待于进一步深入探讨。对于选取一项新技术存在着很大的风险。虽然 AspectJ 对 AOP 的具体实现已经较成熟，但仍存在诸如过于复杂、破坏封装、需要专门的 Java 编译器等缺点。希望在未来的发展中，能够把 AspectJ 集成在新的开发环境中，解决存在的缺点，并弥补自身的不足。

参考文献

1 Lee AH, Zachary JL. Reflections on Metapro gramming. IEEE Trans. on Software Engineering,1995,21(11).

2 Developer Works.AspectJ 和模仿对象的测试灵活性, 2002.5.

3 Stiller E, LeBlanc C.基于项目的软件工程.北京:机械工业出版社出版,2002.

4 吴其庆.JAVA 编程思想与实践.北京:冶金工业出版社出版, 2002.

5 殷人昆,田金兰,马晓勤.实用面向对象软件工程教程.北京: 电子工业出版社,2000.

6 袁望洪,陈向葵,谢涛,郭耀.逆向工程研究与发展.计算机科学,1999,26(5):71－77.

7 Pratt TW, Zelkowitz MV.傅育熙,等译.程序设计语言—设计与实现.第 4 版.北京:电子工业出版社,2001.

8 Steinmetz R, Nahrstedt K. Multimedia. Computing, Communications and Application. Upper Saddle River. NJ: Prentice Hall, 1995.

9 李青山,陈平,王伟,宋海鸿.逆向工程中反射植入的研究.计算机学报,2004,4.

10 龚晓洁.逆向工程中动静态结合分析目标系统的研究[硕士学位论文].西安:西安电子科技大学,2007.

11 http://www.csdn.net/magazine

12 http://www.aspectj.org