International Journal of Control and Automation
Vol.7, No.6 (2014), pp.311-324
http://dx.doi.org/10.14257/ijca.2014.7.6.29

Research on Aspect J Capturing Information of Boolean or Compound Expression-based and Process-Control Join points

Su Ma

*College of Computer Science, Xi'an Polytechnic University
Xi'an, 710048, P.R. China*

31763043@qq.com

Abstract

This As to the research on AspectJ in this context, after analyzing the Java program, AspectJ can realize the capture of the join points based on the Boolean or Compound expression and Process-Control in the Java program. The information includes: the time to judge the runtime condition of each join point as "true"; use of the logic AND (&&) to combine pointcuts; and use of the logic OR (||) to combine pointcuts; declaration of anonymous pointcut and reuse of pointcut;It also includes all join points from beginning in program control-flow and those in which initial one is not included.

Keywords: *AspectJ, Boolean, Compound Expression, Process-Control, Join point, Capture*

1. Introduction

With the development of the computer software development technology, AOP emerges and becomes a better software developing method. The AOP development environment, AspectJ Tool, provides a set of grammars to clearly describe the crosscutting concerns and insert them into the Java source code [1]. The contents in this text is about AspectJ, which, after analyzing the Java program, can realize capturing the join points based on the Boolean or Compound expression and Process-Control in the Java program. The information includes: the time to judge the runtime condition of each join point as "true"; use of the logic AND (&&) to combine pointcuts; and use of the logic OR (||) to combine pointcuts; declaration of anonymous pointcut and reuse of pointcut; It also includes all join points from beginning in program control-flow and those in which initial one is not included. All these mentioned above are the dynamic information created when the program is running, and all relations created during this running will be recorded.

2. Aspect J

AspectJ, supporting the reverse engineering, is a tool for adding such new concepts as Join point, Pointcut, Advice, Inter-type declaration and Aspect into Java program. join point refers to one designated point in the program stream, just like the statements in a program. Pointcut is for collecting the specific set of join points and the values in these join points. Notice refers to the code run once the program has reached any join point. Pointcut and Notice, as the dynamic portion of AspectJ, affect the program flow dynamically. Inter-type declaration affects the program's class structure statically. Aspect is the modular unit of crosscutting concern, which is used to package all these new structures [2]. AspectJ reflects the source program through the defined Aspect, utilizing the ajc compiler to compile Classes and Aspect codes, whereby the inserted software trigger is reflected to the source program through Aspect.

3. Capture of Join Points Based on Boolean or Compound Expression

Considering the way to combine pointcuts, the pointcut is expressed as a Boolean expression and determine whether the specific join point has been captured by judging the pointcut logic. Due to the Boolean character of the pointcut logic, it is allowed to use the traditional Boolean expression, such as the logic AND, OR or NOT, to combine pointcut declarations. As to pointcut declarations, conditional logics can also be expressed by the if-statement, which is for comparing the expressions containing non-Boolean values. During runtime, the program passes the value judged by the if-statement and gives a result of true or false.

3.1. Capturing when the Runtime Condition of the Join Point can be Judged as True

When comparing some runtime values that can be judged at the captured join points, you can use the if(Expression)-statement to judge the Boolean expressions containing the runtime variables to be compared and thereby trigger the true-based notices. Here are the main characters of the if(Expression)-statement: 1. It judges the variables provided at the runtime and produces the result (true or false) to decide whether to trigger the corresponding notice. 2. Exp.1 can be composed of several logic elements, including the demonstrative join point environment, static variable and other pointcut declarations. The following Exp.1 demonstrates the usage of the if(Expression)-statement. If the runtime value of the variable forming an expression produces the result of true, only the after notice will be run.

- **Experiment 1: The if(Expression)-Statement for Judging the Target Application Variables.**

1) Business Logic Java Class

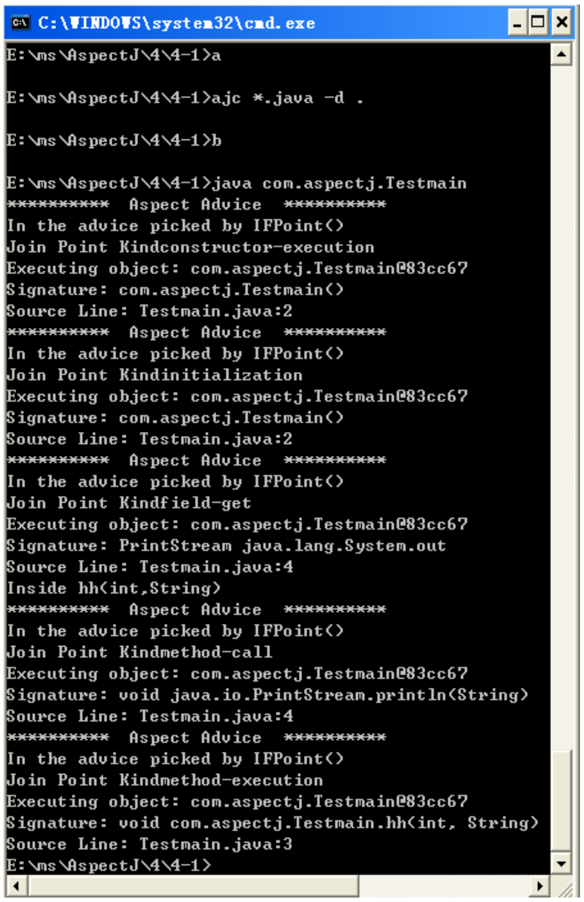
```
package com.aspectj;
  public class Testmain{
    public void hh(int number,String name){
       System.out.println("Inside hh(int,String)");
    }
    public static void main(String[] args){
        Testmain myObject = new Testmain ();
        myObject.hh(4, "M Zhang");
    }
}
```

2) IFRecipe Aspect of AspectJ

```
package com.aspectj;
public aspect IFRecipe{
       private static final long rSalary=20120201;
       pointcut IFPoint():if((thisJoinPoint.getThis() instanceof Testmain)
          && ((Testmain) thisJoinPoint.getThis()).getSalary() < rSalary
```

```
                && ((Testmain) thisJoinPoint.getThis()).getSalary() > 0
                && !withincode(* Testmain.get*())
                && !execution(* Testmain.get*()));
            after(): IFPoint () && !within(IFRecipe+){
              System.out.println("**********  Aspect Advice  **********");
              System.out.println("In the advice picked by IFPoint()");
              System.out.println("Join Point Kind" + thisJoinPoint.getKind());
              System.out.println("Executing object: " + thisJoinPoint.getThis());
              System.out.println("Testmain          instance:          "          +          ((Testmain)
thisJoinPoint.getThis()).getName() + ":" + ((Testmain) thisJoinPoint.getThis()).getSalary());
              System.out.println("Signature: " + thisJoinPoint.getStaticPart().getSignature());
              System.out.println("Source                    Line:                    "                    +
thisJoinPoint.getStaticPart().getSourceLocation());
            }
        }
```

After saving those two files under the same directory and running the ajc command for compiling the files, the files of the aspect and class with suffix .class, that is, Testmain.class and IFRecipe.class, will be created. By running the java command, the following aspect results can be obtained, as shown in Figure 1:



```
E:\ms\AspectJ\4\4-1>a

E:\ms\AspectJ\4\4-1>ajc *.java -d .

E:\ms\AspectJ\4\4-1>b

E:\ms\AspectJ\4\4-1>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice picked by IFPoint()
Join Point Kindconstructor-execution
Executing object: com.aspectj.Testmain@83cc67
Signature: com.aspectj.Testmain()
Source Line: Testmain.java:2
**********  Aspect Advice  **********
In the advice picked by IFPoint()
Join Point Kindinitialization
Executing object: com.aspectj.Testmain@83cc67
Signature: com.aspectj.Testmain()
Source Line: Testmain.java:2
**********  Aspect Advice  **********
In the advice picked by IFPoint()
Join Point Kindfield-get
Executing object: com.aspectj.Testmain@83cc67
Signature: PrintStream java.lang.System.out
Source Line: Testmain.java:4
Inside hh(int,String)
**********  Aspect Advice  **********
In the advice picked by IFPoint()
Join Point Kindmethod-call
Executing object: com.aspectj.Testmain@83cc67
Signature: void java.io.PrintStream.println(String)
Source Line: Testmain.java:4
**********  Aspect Advice  **********
In the advice picked by IFPoint()
Join Point Kindmethod-execution
Executing object: com.aspectj.Testmain@83cc67
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:3
E:\ms\AspectJ\4\4-1>
```

Figure 1. Program Running Result of Experiment 1

In the above IFRecipe aspect, line 2 declares an aspect while line 4 declares the pointcut logic of a single name. The pointcut logic designates the join points of the application. In this example, it captures the variables provided through the pointcut judgment at the runtime, and produces the result (true or false) about whether to trigger the corresponding notice. The pointcut is named as IFPoint (), through which the pointcut can be called within the whole aspect action scope. Line 9 to 20 declares a single notice block. The after () notice only simply indicates that it will be run after any join point matched to the IFPoint pointcut.

Note: in this example, the conditional logic included in the IFPoint () pointcut refers to the following events: , the corresponding after notice should be triggered when the running object is of the Testmain type, the object attribute salary is lower than the constant rSalary but higher than 0, or the current join point is not called in the getSalary() method. Through wildcard character, no join point is allowed to appear in any method of the Testmain class reached firstly.

3.2. Use of Logic and (&&) to Combine Pointcuts

If you want to combine some pointcut declarations for realizing that as long as all the conditions within the pointcut declarations are judged as true, the notice on the join point will be run, you can use the && operator. The following Exp.2 shows how to use the && operator, which combines two pointcut logics into one pointcut declaration. Figure 2 demonstrates how to use the && operator to combine two pointcuts.

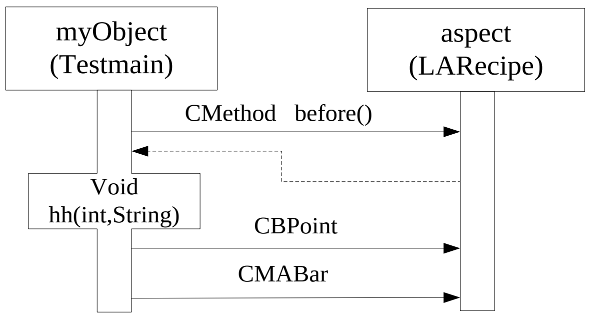
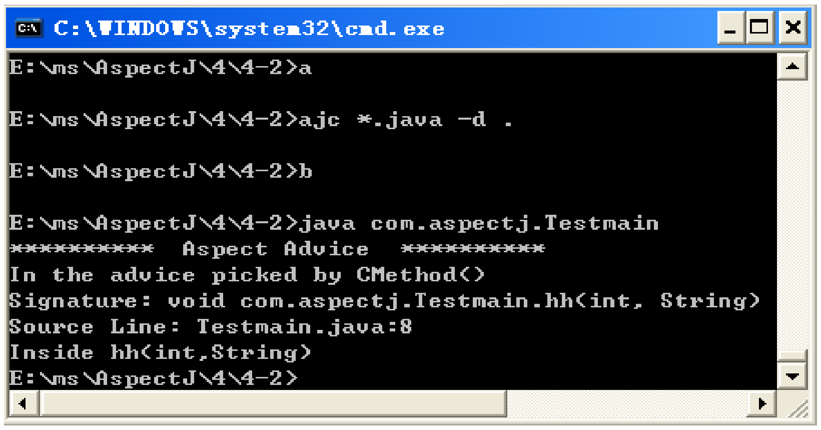


Figure 2. How to Use the && Operator to Combine Two Pointcuts

- **Experiment 2: Use of the && Operator to Combine Two Pointcuts.**

```
package com.aspectj;
  public aspect LARecipe{
    pointcut CMethod():call(* Testmain.* (..));
    pointcut CBPoint():call(void Testmain.bar());
    pointcut CMABar():CMethod() && CBPoint();
    before():CMethod ()&&!within(LARecipe+){
      System.out.println("**********  Aspect Advice  **********");
      System.out.println("In the advice picked by CMethod()");
      System.out.println("Signature: " + thisJoinPoint.getSignature());
      System.out.println("Source Line: " + thisJoinPoint.getSourceLocation());
    }
    before(): CBPoint()&&!within(LARecipe+){
        System.out.println("Signature: " + thisJoinPoint.getSignature());
    }
    before(): CMABar ()&&!within(LARecipe+){
        System.out.println("Source Line: " + thisJoinPoint.getSourceLocation());
    }
}
```

The aspect's running result is as shown in Figure 3:



```
E:\ms\AspectJ\4\4-2>a

E:\ms\AspectJ\4\4-2>ajc *.java -d .

E:\ms\AspectJ\4\4-2>b

E:\ms\AspectJ\4\4-2>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice picked by CMethod()
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
Inside hh(int,String)
E:\ms\AspectJ\4\4-2>
```

Figure 3. Program Running Result of Experiment 2

In the AspectJ environment, when two or more simple pointcuts are combined as one compound pointcut with the && operator, the join points selected by two separated pointcuts will trigger the notice of the compound pointcut [3]. The sequence of combining pointcuts with the && operator may also affect the interpretive mode of the compound pointcut. Runtime analysis for the && operator is from left to right, which means that when checking the candidate join points, the first pointcut with no join point is the position to end the comparison.

3.3. Use of Logic OR(||) to Combine Pointcuts

If you want to combine some pointcut declarations for running the notice on the join point as long as one of the conditions within the pointcut declarations is judged as true, it is recommended to use the || operator [4]. The following Exp.3 shows how to use the || operator, which combines the pointcuts CHPoint() and CBPoint() into one compound pointcut named as CHOBar(). Figure 4 demonstrates how to use the || operator to combine two pointcuts.

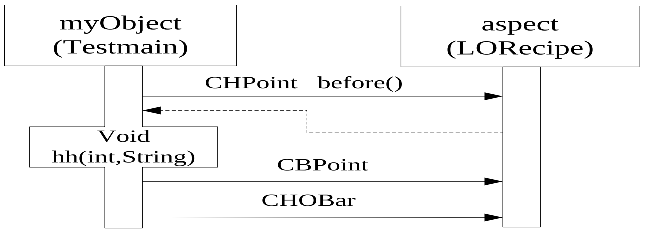
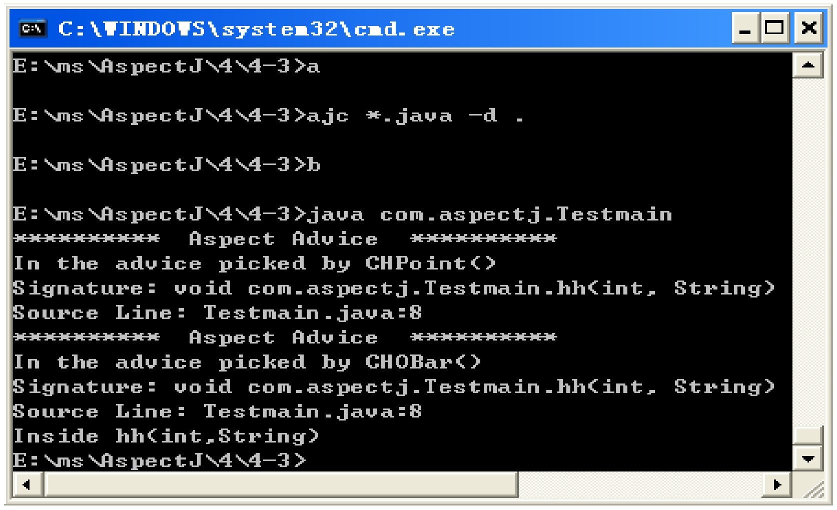


Figure 4. How to Use the || Operator to Combine Two Pointcuts

- **Experiment 3: Use of the || Operator to Combine Two Pointcut Declarations.**

```
package com.aspectj;
 public aspect LORecipe{
    pointcut CHPoint(): call(void Testmain.hh(int,String));
    pointcut CBPoint() : call(void Testmain.bar());
    pointcut CHOBar(): CHPoint() || CBPoint();
    before(): CHPoint()&&!within(LORecipe+){
      System.out.println("**********  Aspect Advice  **********");
      System.out.println("In the advice picked by CHPoint()");
      System.out.println("Signature: " + thisJoinPoint.getSignature());
      System.out.println("Source Line: " + thisJoinPoint.getSourceLocation());
    }
    before():CBPoint()&&!within(LORecipe+){
      System.out.println("Signature: " + thisJoinPoint.getSignature());
    }
    before(): CHOBar()&&!within(LORecipe+){
      System.out.println("**********  Aspect Advice  **********");
      System.out.println("In the advice picked by CHOBar()");
      System.out.println("Signature: " + thisJoinPoint.getSignature());
      System.out.println("Source Line: " + thisJoinPoint.getSourceLocation());
     }
 }
```

The Aspect's Running Result is as Shown in Figure 5:



```
E:\ms\AspectJ\4\4-3>a

E:\ms\AspectJ\4\4-3>ajc *.java -d .

E:\ms\AspectJ\4\4-3>b

E:\ms\AspectJ\4\4-3>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice picked by CHPoint()
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
**********  Aspect Advice  **********
In the advice picked by CHOBar()
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
Inside hh(int,String)
E:\ms\AspectJ\4\4-3>
```

Figure 5. Program Running Result of Experiment 3

In the AspectJ language, if any join point, with the help of the logic OR, is combined as one compound pointcut, the join point will trigger the notice after starting the notice of the pointcut. The || operator in AspectJ also demonstrates the short circuit action, whose short circuit rule is opposite to the && operator. With the help of the || operator, at the moment when one condition is judged as true, the compound statement will also be judged as true.

3.4. Declaration of Anonymous Pointcut

If you want to declare a simple pointcut in the named pointcut declaration anonymously, or add the pointcut to a notice, you can declare an anonymous pointcut [5]. Anonymous pointcut is a member of pointcut declaration, which will be used in all pointcut-based parts. The following Exp.4 is about an anonymous pointcut, which can be used as a basis of a more complicatedly named pointcut or for notice declaration directly. Figure 6 demonstrates how to use the anonymous pointcut.

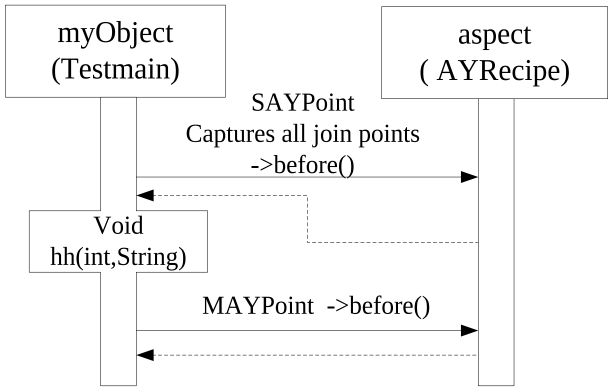
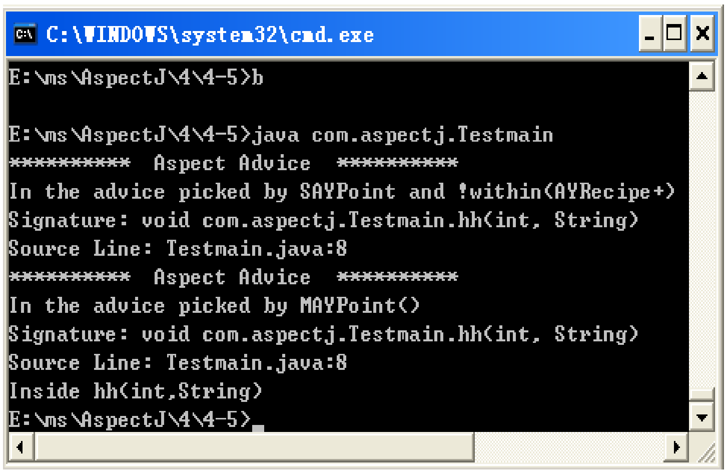


Figure 6. How to Use Anonymous Pointcut

- **Experiment 4: Use of Anonymous Pointcut.**

```
package com.aspectj;
 public aspect AYRecipe{
   pointcut SAYPoint(): call(void Testmain.hh(int,String));
   pointcut MAYPoint(): (call(void Testmain.bar())
        || call(void Testmain.hh(int,String))
        && !within(AYRecipe+));
   before(): SAYPoint()&&!within(LNRecipe+){
    System.out.println("**********  Aspect Advice  **********");
    System.out.println("In the advice picked by SAYPoint and !within(AYRecipe+)");
    System.out.println("Signature: " + thisJoinPoint.getSignature());
    System.out.println("Source Line: " + thisJoinPoint.getSourceLocation());
   }
   before(): MAYPoint(){
     System.out.println("**********  Aspect Advice  **********");
     System.out.println("In the advice picked by MAYPoint()");
     System.out.println("Signature: " + thisJoinPoint.getSignature());
     System.out.println("Source Line: " + thisJoinPoint.getSourceLocation());
   }
 }
```

The aspect's running result is as shown in Figure7:



```
E:\ms\AspectJ\4\4-5>b

E:\ms\AspectJ\4\4-5>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice picked by SAYPoint and !within(AYRecipe+)
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
**********  Aspect Advice  **********
In the advice picked by MAYPoint()
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
Inside hh(int,String)
E:\ms\AspectJ\4\4-5>_
```

Figure 7. Program Running Result of Experiment 4

Anonymous pointcut is for declaring pointcuts rather than checking some specific pointcut types. You can declare pointcuts within any named pointcut declaration anonymously or by adding the pointcuts directly to the notice they called. Anonymous pointcut can be used as a member of any more complicatedly named pointcut declaration, that is, it can be declared as part of a compound-named pointcut or part of a notice declaration. In this way, the named pointcut may reversely be a member of the reused pointcut logic.

3.5. Reuse of Pointcut

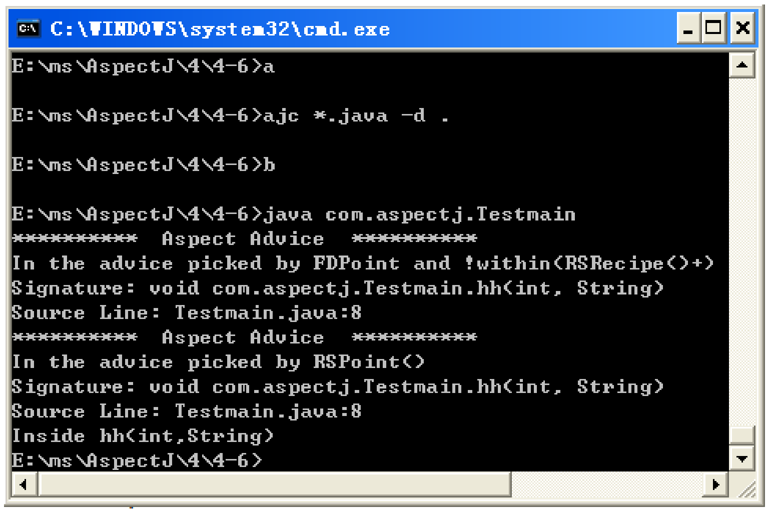
If you want to reuse a pointcut expression, you can declare a pointcut, and call the pointcut through its name at the positions necessary to reuse the pointcut expression. The following Exp.5 is about reuse of a named pointcut, in which the named FDPoint() pointcut is reused when RSPoint() is declared.

- **Experiment 5: Reuse of a Named Pointcut.**

```
package com.aspectj;
 public aspect RSRecipe{
   pointcut FDPoint(): call(void Testmain.hh(int,String));
   pointcut RSPoint(): FDPoint()&&!within(RSRecipe+);
   before(): FDPoint()&&!within(RSRecipe+){
     System.out.println("**********  Aspect Advice  **********");
     System.out.println("In the advice picked by FDPoint and !within(RSRecipe()+)");
     System.out.println("Signature: " + thisJoinPoint.getSignature());
     System.out.println("Source Line: " + thisJoinPoint.getSourceLocation());
   }
   before(): RSPoint(){
      System.out.println("**********  Aspect Advice  **********");
      System.out.println("In the advice picked by RSPoint()");
      System.out.println("Signature: " + thisJoinPoint.getSignature());
      System.out.println("Source Line: " + thisJoinPoint.getSourceLocation());
```

```
    }
}
```

The Aspect's Running Result is as Shown in Figure 8:


```
E:\ms\AspectJ\4\4-6>a

E:\ms\AspectJ\4\4-6>ajc *.java -d .

E:\ms\AspectJ\4\4-6>b

E:\ms\AspectJ\4\4-6>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice picked by FDPoint and !within(RSRecipe()+)
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
**********  Aspect Advice  **********
In the advice picked by RSPoint()
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
Inside hh(int,String)
E:\ms\AspectJ\4\4-6>
```

Figure 8. Program Running Result of Experiment 5

Pointcut naming is similar to a method, and has a signature. The named pointcut can be called by all the parts below the pointcut declaration within a specific method. Considering the inheritance relationship of each aspect, you can declare the pointcut as abstract and realize the actual pointcut logic through the inheritance aspect. When you declare a pointcut for reusing, the position where you reuse the pointcut declaration is critical [6]. In AspectJ, the pointcut declaration has the same access modifier with common Java methods: the public pointcut declaration is visible within the whole application aspect; the default pointcut declaration is visible for all the other aspects within the same package; the protected pointcut declaration is only visible for sub-aspects; and the private pointcut declaration is only visible within the aspect including this pointcut declaration.

4. Capture Join Points Based on Flow-Control

Each join point has a specific location in the program's flow, and this provides an environment for capturing join points by the pointcut declaration. Image join point as a specific point in codes, and it provides a tag for cope definition which pointcut will select. Pointcut described in the following supports to capture all join points in another initial join point scope or enviroment.

4.1. Capture All Join Points Passing the Beginning of Initial Join Points in Program Flow-Control

If you want to capture all join points that are encountered in the program control-flow, and they are all after the beginning of the selected join point through a single pointcut including pointcut itself, you may use “cflow(Pointcut)”. Its main characteristics [7] are: it can capture all the join points encountered in the initial join point environment which is selected through another pointcut. join points captured include the initial one, the scope is its important differential, and it can capture all join points by pointcut parameters of the control-flow. Exp.6 below illustrates how “cflow (Pointcut)” capture all join points in program control-flow. These join points are all after the initial one captured by “CIPoint()”. Figure9 illustrates how to use “cflow (Pointcut)” pointcut.

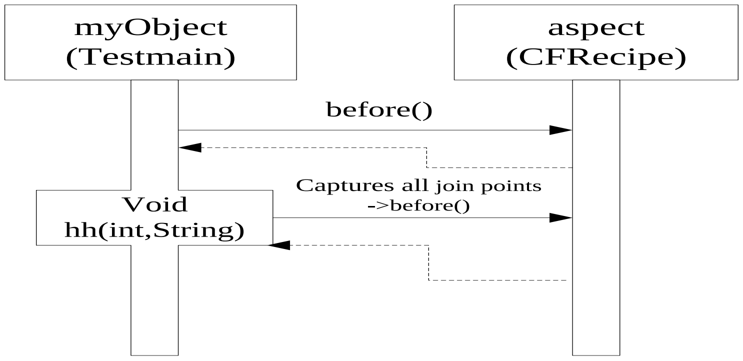
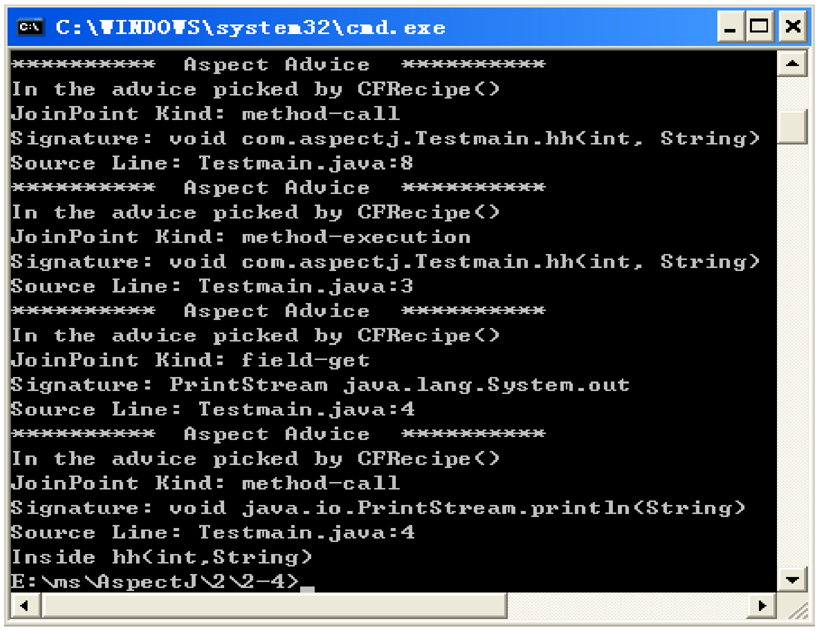


Figure 9. How to Use “Cflow (Pointcut)” Pointcut

- **Experiment 6: Using “Cflow (Pointcut)” Capture All Join Points After and Including Special Pointcut**

```
package com.aspectj;
 public aspect CFRecipe{
   pointcut CIPoint():call(void Testmain.hh(int,String));
   pointcut CFPoint():cflow(CIPoint());
   before(): CFPoint() && !within(CFRecipe+){
            System.out.println("**********  Aspect Advice  **********");
            System.out.println("In the advice picked by CFRecipe()");
            System.out.println("JoinPoint Kind: " + thisJoinPoint.getKind());
            System.out.println("Signature:                    "                    +
   thisJoinPoint.getStaticPart().getSignature());
            System.out.println("Source              Line:              "              +
   thisJoinPoint.getStaticPart().getSourceLocation());
   }
 }
```

Program Running Result is Shown in Figure 10:



```
***********  Aspect Advice  ***********
In the advice picked by CFRecipe()
JoinPoint Kind: method-call
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
***********  Aspect Advice  ***********
In the advice picked by CFRecipe()
JoinPoint Kind: method-execution
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:3
***********  Aspect Advice  ***********
In the advice picked by CFRecipe()
JoinPoint Kind: field-get
Signature: PrintStream java.lang.System.out
Source Line: Testmain.java:4
***********  Aspect Advice  ***********
In the advice picked by CFRecipe()
JoinPoint Kind: method-call
Signature: void java.io.PrintStream.println(String)
Source Line: Testmain.java:4
Inside hh(int,String)
E:\ms\AspectJ\2\2-4>
```

Figure 10. Program Running Result of Experiment 6

“cflow (Pointcut)” pointcut introduces the concept of the join point environment. It refers to the scope of every join point in which it is seen as part of the implementation of the program control-flow [8]. In this control process, any connection point encountered will trigger the associated notifications. Then, it calls the relating notification for every connection point in the scope of control-flow. At last, the biggest difference between this pointcut and “cflowbelow(Pointcut)” one is that join points captured include the initial one [9]. Implementation of “Cflow (Pointcut)” needs a lot of system resources, so “withincode(Signature)” but not “cflow(Pointcut)” is preferred when join -point-reuse is not affected.

4.2. Capture All Join Points of Program Control-Flow Not Including the Initial One

If you want to capture all join points that are encountered in the program control-flow, and they are after the beginning of the selected one by pointcut, you may use “cflowbelow (Pointcut)” pointcut [10]. The following description illustrates capturing all the join points which are after the one that is captured by “CIPoint()” pointcut. Figure 11 illustrates how to apply “cflowbelow (Pointcut)” pointcut.

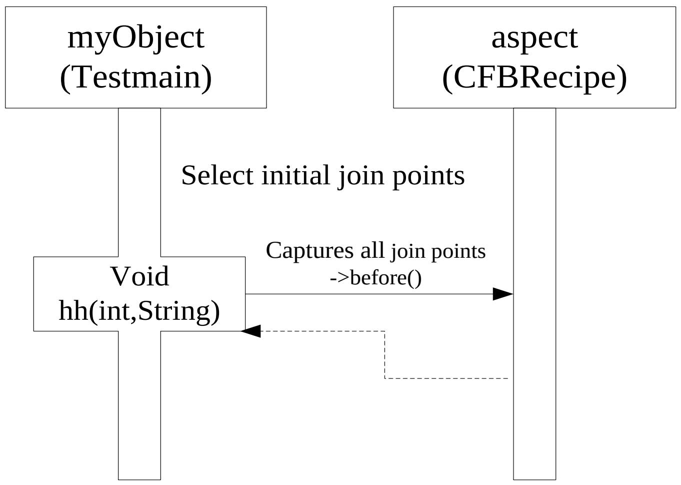
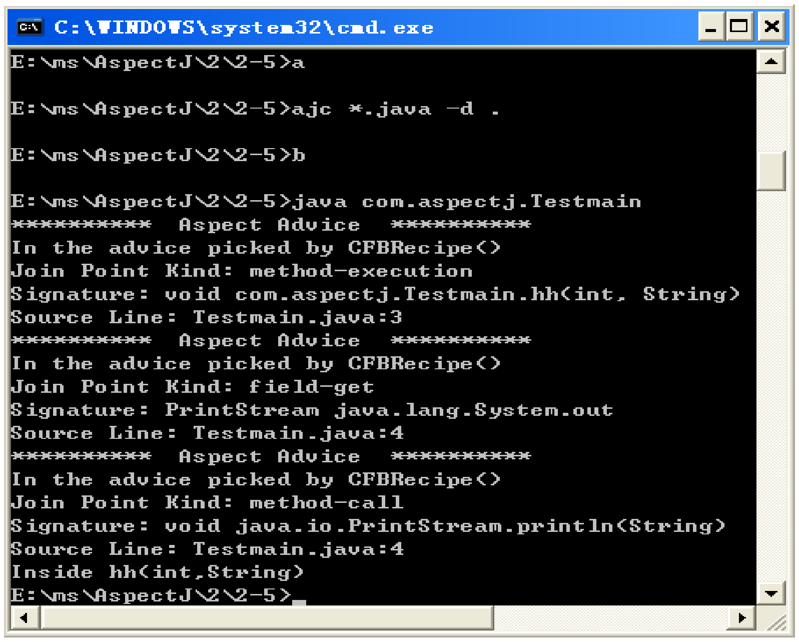


Figure 11. How to Apply "cflowbelow (Pointcut)" Pointcut

- **Experiment 7: Use "cflowbelow(Pointcut)" Pointcut Capture All Join Points After Initial One**

```
package com.aspectj;
  public aspect CFBRecipe{
     pointcut CIPoint():call(void Testmain.hh(int,String));
     pointcut CFPoint():cflowbelow(CIPoint());
     before(): CFPoint() && !within(CFBRecipe+){
                   System.out.println("**********  Aspect Advice  **********");
                   System.out.println("In the advice picked by CFBRecipe()");
                   System.out.println("Join Point Kind: " + thisJoinPoint.getKind());
                   System.out.println("Signature:                    "                    +
     thisJoinPoint.getStaticPart().getSignature());
                   System.out.println("Source              Line:             "            +
     thisJoinPoint.getStaticPart().getSourceLocation());
     }
  }
```

Figure 12 Shows Program Executing Result:



```
E:\ms\AspectJ\2\2-5>a

E:\ms\AspectJ\2\2-5>ajc *.java -d .

E:\ms\AspectJ\2\2-5>b

E:\ms\AspectJ\2\2-5>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice picked by CFBRecipe()
Join Point Kind: method-execution
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:3
**********  Aspect Advice  **********
In the advice picked by CFBRecipe()
Join Point Kind: field-get
Signature: PrintStream java.lang.System.out
Source Line: Testmain.java:4
**********  Aspect Advice  **********
In the advice picked by CFBRecipe()
Join Point Kind: method-call
Signature: void java.io.PrintStream.println(String)
Source Line: Testmain.java:4
Inside hh(int,String)
E:\ms\AspectJ\2\2-5>_
```

Figure 12. Program Executing Result of Experiment 7

Here, join -point number is different with that in section IV.A. “cflow(Pointcut)” pointcuts trigger the notification of all join points in initial environment while “cflowbelow(Pointcut)” does not trigger that of the initial one.

5. Conclusion

Through the research on AspectJ capturing join points based on Boolean or compound expression, it successfully realizes the capture of the information about the join points based on the Boolean or Compound expression and Process-Control in the Java program. AspectJ is an extension of Java AOP whose aspects can be run in all development phases in case that the original code has not been changed. Although AspectJ has more mature AOP technology, defects still exist, for example, it may damage packaging and require special Java compiler. Currently, the research on AspectJ development is still at the primary exploratory stage, and needs to be further discussed.

I hereby certify that, the principal views and contents in this text come from AspectJ Cookbook compiled by Russ Miles. On that basis, the running results of all examples are provided with specific explanations, which are the main attraction for readers, so as to give them a deeper impression.

Acknowledgements

The author wishes to express her gratitude to the referees and editors for their valuable comments on earlier draft of the paper.

References

[1] M. N.Malta and M. T. D. O. Valente, “Object-oriented transformations for extracting aspects”, Information and Software Technology, vol. 51, no. 1, **(2009)**, pp. 138-149.

[2] Z. Hui, Z. Yang, G. -Q. Lu and X. -L. Zhou, “Design and implementation of library management system based on AOP”, Hebei Journal of Industrial Science and Technology, vol. 26, no. 5, **(2009)**.
[3] C. Ping and X. Min, “A Java Contractual Programming Language Model Based on Aspect,” Jour Northeast Dianli University, vol. 31, no. 1, **(2011)**.
[4] M. Su, Q. -S. Li and P. -G. Chen, “Basic Information Needed in Reversing-Engineer by AspectJ”, Computer Systems & Applications, vol. 20, no. 2, **(2011)**.
[5] Y. Yin and J. -D. Zhang, “Analysis of the Aspect Abnormal Type of Fault”, Computer Programming Skills & Maintenance, vol. 21, no. 1, **(2010)**.
[6] N. -S. Yao, “Java Programming”, Wuhan University, Wuhan, **(2006)**.
[7] R. Miles, “Aspect Cookbook, Edited Chang Xiao-Bo”, Tsinghua University, Beijing, vol. 1, **(2006)**, pp. 104-109.
[8] R. Miles, “Aspect Cookbook, Edited Chang Xiao-Bo”, Tsinghua University, Beijing, vol. 1, **(2006)**, pp. 120-133.
[9] Z. X. Yin, “International Conference on Computer Design and Applications”, Xi’an, Shaanxi, China, **(2011)** May 27-29.
[10] J. Y. Liang and P. X. Wu, “International Applied Mechanics”, Mechatronics Automation & System Simulation Meeting, Hangzhou, Zhejiang, China, **(2012)** June 24-26.

Author

Su Ma

She comes from Xi'an, Shaanxi province. She received Bachelor degree from Shaanxi University of Science & Technology in 2001 and Master degree from Xi'an University of Electronic Science and Technology in 2005.Since 2001, she has been a teacher in Department of Computer Science of Xi'an Polytechnic University in Xi'an, Shaanxi province, China. She engaged in the research on Software Engineering and Intelligent-Control. She has published more than 15 first-author pagers and a scholarly monograph. She has been responsible for a provincial-level research project.