International Journal of Hybrid Information Technology
Vol.7, No.4 (2014), pp.55-68
http://dx.doi.org/10.14257/ijhit.2014.7.4.06

# A Research on Join Points-Capture in Method and Object by AspectJ

Su Ma

*College of Computer Science, Xi'an Polytechnic University*
*Xi'an, 710048, P.R. China*
*31763043@qq.com*

***Abstract***

*This paper achieves information-capture of join points by analyzing Java program with AspectJ. This information includes: method call, parameter values passed in method call, captured value of reference "this" when method is executed, the time when this references a specific type problem, and the time when the target object of the join point is a specific type.*

***Keywords****: Java, AspectJ, Capture, Method, Object, Join point*

## 1. Introduction

With the development of computer software technology, Aspect-Oriented Programming (AOP) will be a good way to write software, it can separate the concerns of cross-section of multi-module software. AOP developing environment, the AspectJ tools, is an extension of Java language, it provides a set of syntax and is able to describe clearly the crosscutting concerns and implant that into the Java source codes [1]. This article achieves information-capture of join-point by analyzing Java program with AspectJ. This information includes: method call, parameter values passed in method call, captured value of reference "this" when method is executed, the time when this references a specific type problem, and the time when the target object of the join point is a specific type. This information is created dynamically in program which records various relationships in runtime.

## 2. AspectJ

AspectJ adds the concepts of join point, pointcut, advice, inter-type declaration and aspect into Jave without which traditional programming way cannot suppose inverse engineer well. Pointcut and advice affect dynamically program flow, inter-type declarations affect static class-structure of the program, and aspect is a package of all these new structures. AspectJ itself cannot suppose inverse engineering, but it can reflect the resource program by the defined aspects, compile the classes and aspect codes by "ajc" tool and generate a valid class or java files [2], which can be reflected into initial program by aspects.

## 3. Capture Method Call

Join point is a specific point at which advice can be implanted in the application code. The pointcut provides a logical definition for the join point selection of advice call. Exp.1 to 6, cited in this article, correspond to the construct-type of Java language. The file is named with "Testmain.java" which contains the join point of pointcut declaration in AspectJ.

If you want to capture the specific method matching signature, you can use "call(Signature)" pointcut. The "call(Signature)" pointcut has two key features [3]: 1. It triggers advice when method is called, and the environment is calling class. 2. "Signature" may contain wildcards for selecting a series of join points on the different classes and methods. For example, "* void Testmain.hh, (int, float)", regardless of wildcards, join point of method will be captured. This can also be achieved by ignoring the visibility of the modifier. Regardless of the modifier, return type, class, or any number of parameters, join point of method will be captured.

The following declares a "call(Signature)" pointcut in method which matches the "com.aspectj.Testmain.hh(int String)" signature. Figure 1 illustrates how to use "call (Signature)" pointcut in a simple class.

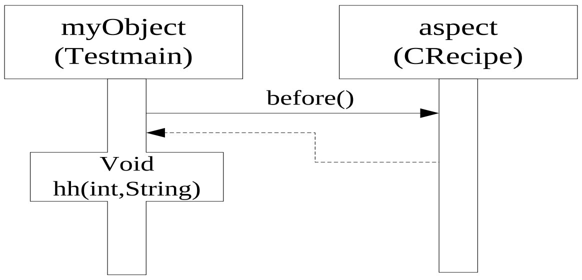


**Figure 1. How to Use "Call(Signature)" Pointcut**

**Exp. 1: Use "call(Signature)" Pointcut to Capture the Call of Specific Method Signature.**

1) Business Logic Java Class

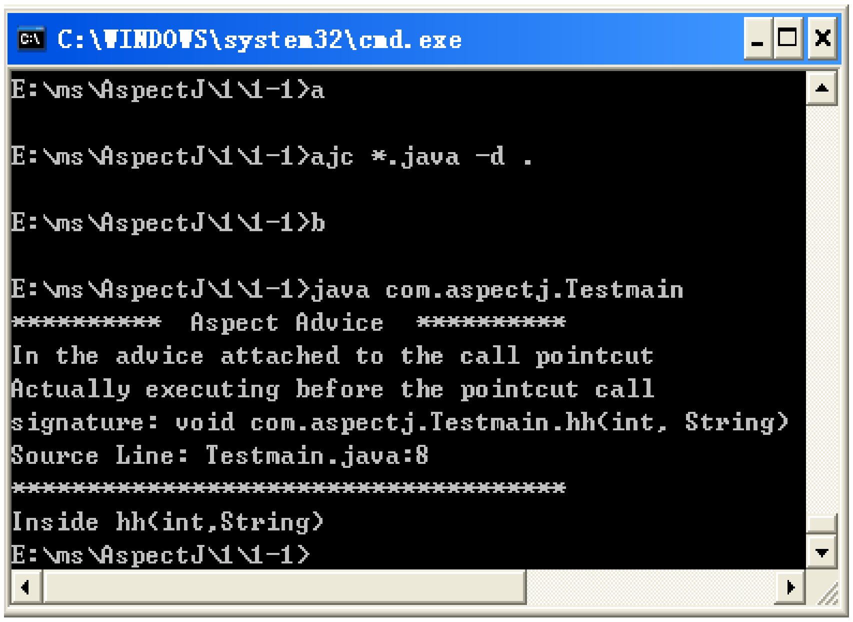
```
package com.aspectj;
  public class Testmain{
      public void hh(int number,String name){
          System.out.println("Inside hh(int,String)");
      }
      public static void main(String[] args){
          Testmain myObject = new Testmain ();
          myObject.hh(1, "M Zhang");
      }
  }
```

2) "CRecipe" aspect of AspectJ

```
package com.aspectj;
public aspect CRecipe{
    pointcut cPoint(): call(void  com.aspectj.Testmain.hh(int, String));
   //Advice declaration
    before(): cPoint(){
           System.out.println("**********  Aspect Advice  **********");
           System.out.println("In the advice attached to the call pointcut");
           System.out.println("Actually executing before the pointcut call");
           System.out.println("signature:                    "                    +
thisJoinPoint.getStaticPart().getSignature());
```

```
                System.out.println("Source                Line:               "               +
                thisJoinPoint.getStaticPart().getSourceLocation());
                System.out.println("************************************");
            }
        }
```

Save these two files in the same directory, and run command "ajc" to compile these two files. Aspect and Class files (Testmain.class and CRecipe.class) are generated. Run by java command, following result of the aspect is shown in Figure 2:



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

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

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

E:\ms\AspectJ\1\1-1>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice attached to the call pointcut
Actually executing before the pointcut call
signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
*************************************
Inside hh(int,String)
E:\ms\AspectJ\1\1-1>
```

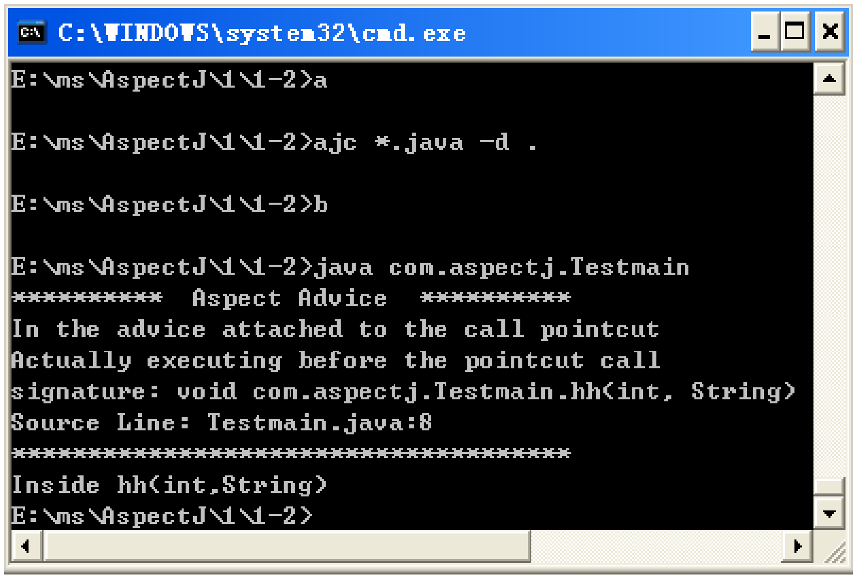
**Figure 2. Program Running Result of Exp.1**

In above, the "Crecipe" aspects, Line 2, declares an aspect. Line 3 to 4 declares the logic of single-name pointcut. The logic of the pointcut specifies any join points of the application. In this case, method call of "void com.aspectj.Testmain.hh(int String)" is captured. The pointcut is named with "cPoint()", and thus it can be referred in any position of aspect scope. Line 6 to19 declare a single advice block. Advice "before ()" is simply sure that it will be executed before any matching join points of the pointcuts.

If the Java class is changed with the following program block:

```
package com.aspectj;
  public class Test extends Testmain{
    public void hh(int age, String name){
       System.out.println("hh(int , String)");
    }
    public static void main(String[] args){
       Test AJ = new Test();
       AJ.hh(34, "M Zhang");
    }
  }
```

Respect Running is shown in Figure 3:



```
E:\ms\AspectJ\1\1-2>a

E:\ms\AspectJ\1\1-2>ajc *.java -d .

E:\ms\AspectJ\1\1-2>b

E:\ms\AspectJ\1\1-2>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice attached to the call pointcut
Actually executing before the pointcut call
signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
*************************************
Inside hh(int,String)
E:\ms\AspectJ\1\1-2>
```

**Figure 3. Program Running Result after Java Class is Changed**

"Test.java" is subclass of "Testmain.java", and rewrites the "hh (int, String)" method. In accordance with the normal methods of implementation in Java, any method is not involved by running "Test.java", but the fact is contrary, "cPoint ()" advice will be executed. This is the special place in AspectJ designing.

Attention: Files "Testmain.java" and "Tess.java" must be compiled at same time in same direction.

## 4. Capture Parameter Values Passed in Method Call

If you want to capture and use the parameters passed in the method call, you can use "call (Signature)" and "args ([TypePatterns | Identifiers])" pointcut to capture method call, then bind the identifier needed to the parameter values of method.

The following is used for a pointcut declaration which matches all "com.aspectj.Testmain.hh (int, String)" signature methods. "CPamaters (int, String)" pointcut needs to specify an integer and string through the value and name of the identifier. Then, through the "args ([the Types | Identifiers])" pointcut bind the identifier to the method parameters. Figure 4 illustrates how to capture parameter values passed on "Testmain.foo (..)" method call.

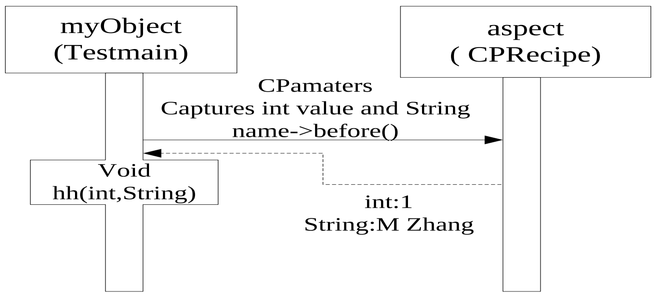
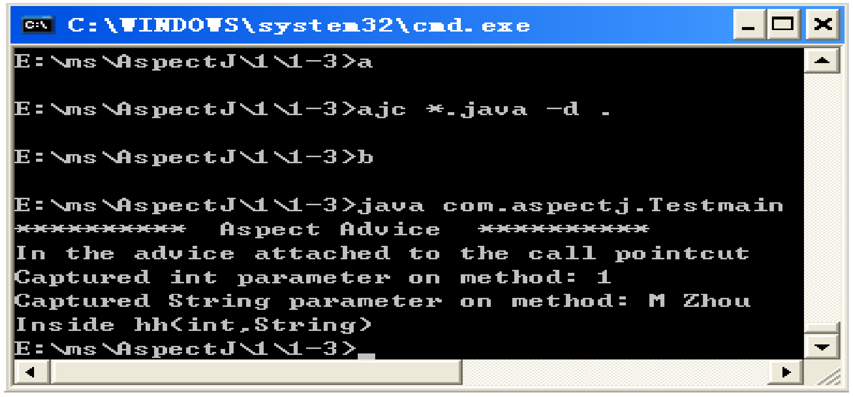


**Figure 4. How to Capture Parameter Values Passed on "Testmain.foo (..)" Method Call**

**Exp. 2: Capture "int" and "String" Value Passed on "Testmain.foo(..)" Method Call.**

```
package com.aspectj;
 public aspect CPRecipe{
   pointcut CPamaters(int value, String name):call(void
              Testmain.hh(int,String))&&args(value, name);
   before(int value,String name):CPamaters(value , name){
     System.out.println("********** Aspect Advice **********");
     System.out.println("In the advice attached to the call pointcut");
     System.out.println("Captured int parameter on method: " + value);
     System.out.println("Captured String parameter on method: " + name);
   }
 }
```

Aspect running result is shown in Figure 5:



```
E:\ms\AspectJ\1\1-3>a

E:\ms\AspectJ\1\1-3>ajc *.java -d .

E:\ms\AspectJ\1\1-3>b

E:\ms\AspectJ\1\1-3>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice attached to the call pointcut
Captured int parameter on method: 1
Captured String parameter on method: M Zhou
Inside hh(int,String)
E:\ms\AspectJ\1\1-3>
```

**Figure 5. Program Running Result of Exp. 2**

There are two binding processes in the above example. In the first time, "Cpamaters" binds "hh" method parameters to own parameters by "args([Types | Identifiers])". In the second time, it binds own parameters to "before()". Therefore, in the pointcut of "CPamaters (int value, String name): call (void Testmain.hh (int, String)) && args (value, name)", all parameters must be corresponded. And, parameters in" before (int value. String name): CPamaters (value, name)" must be corresponded. Parameters between 1 and 2 do not need one-to-one correspondence.

## 5. Capture it when a Method is Executed

If you want to capture it when performing a method which matches a specific signature, you can use the "execution (Signature)" pointcut. "Execution (Signature)" pointcut has two key features [4]: 1. Join-point-triggering environment is in target class method. 2. "Signature" may contain wildcards to select a series of join points on the different classes and methods.

The following is used to declare an "execution(Signature)" pointcut concerned in method join point which matches "com.aspectj.Testmain.hh(int, String)" signature. Figure 6 illustrates how to apply "execution (Signature)" pointcut in a simple class.

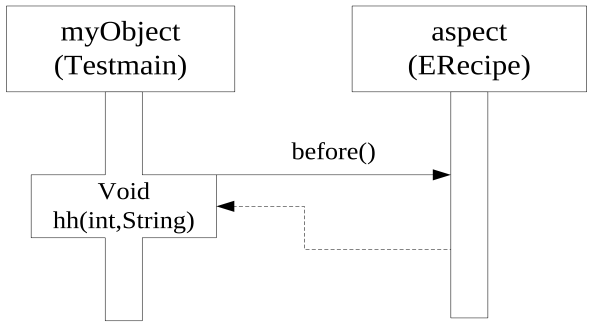


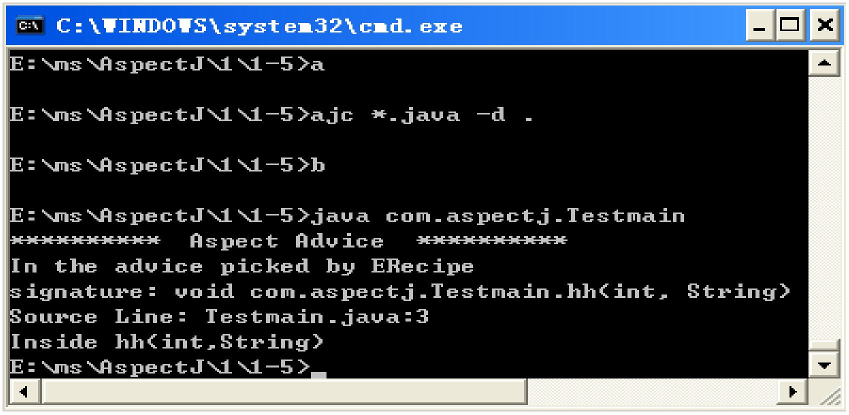
**Figure 6. How to Apply "Execution (Signature)" Pointcut in a Simple Class**

**Exp. 3: Use "execution(Signature)" Method to Capture Join Points in Method Execution.**

```
package com.aspectj;
  public aspect ERecipe{
  pointcut EPoint():execution(void Testmain.hh(int String));
  before(): EPoint() && !within(ERecipe +){
      System.out.println("**********  Aspect Advice  **********");
      System.out.println("In the advice picked by ERecipe");
      System.out.println("signature: " + thisJoinPoint.getStaticPart().getSignature());
      System.out.println("Source Line: " + thisJoinPoint.getStaticPart().getSourceLocation());
  }
 }
```

Comparing with "call(Signature)" pointcut, the above-mentioned codes are not new content. However, please note where the advice is called as well as its environment. Particularly concern the return value of "thisJoinPoint.getStaticPart (). GetSourceLocation ()".

Aspect Running Result is shown in Figure 7:



```
E:\ms\AspectJ\1\1-5>a

E:\ms\AspectJ\1\1-5>ajc *.java -d .

E:\ms\AspectJ\1\1-5>b

E:\ms\AspectJ\1\1-5>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice picked by ERecipe
signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:3
Inside hh(int,String)
E:\ms\AspectJ\1\1-5>_
```

**Figure 7. Program Running Result of Exp. 3**

## 6. Capture the Value Referenced by "this" when Method is Executed

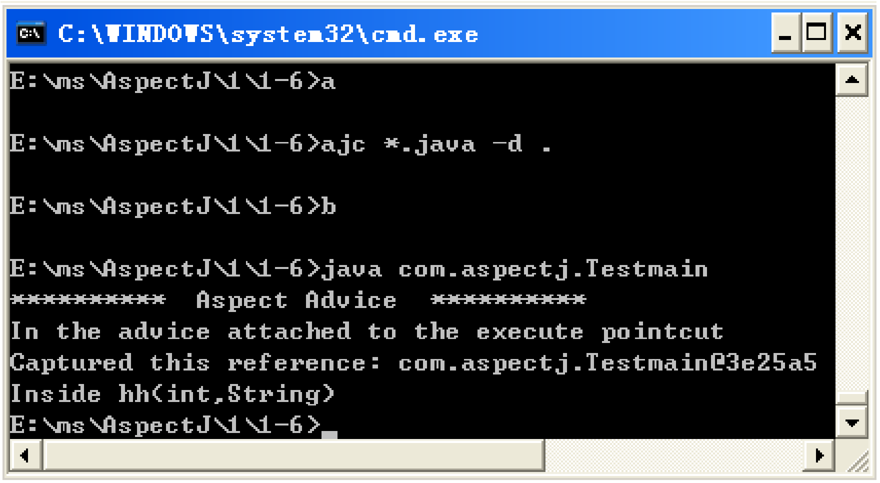
When capturing method during execution, if you want to display Java object referenced by "this" to be used by advice, you can use "execution (Signature)" and "this (Type | Identifier)" pointcuts to capture the implementation of the method, and bind the single identifier to the object referenced by "this" when method is executing [5].

The following declares a concern "execution(Signature)" pointcut in method which matches "com.aspectj.Testmain.hh(int, String)" signature. The "TDExe (Testmain)" pointcut needs a "Testmain" object specified by "myobject" identifier. Then, bind the "myobject" to the method referenced by "this (Type | Identifier)" pointcut.

**Exp. 4: Capture "this" Reference when "Testmain.foo(..)" Method is Executed.**

```
package com.aspectj;
  public aspect TRRecipe{
      pointcut TDExe(Testmain myObject) :
      execution(void Testmain.hh(int, String)) && this (myObject);
      before(Testmain myObject): TDExe(myObject){
             System.out.println("**********  Aspect Advice  **********");
            System.out.println("In the advice attached to the execute pointcut");
            System.out.println("Captured this reference: " + myObject);
     }
  }
```

Aspect Running Result is shown in Figure 8:



```
E:\ms\AspectJ\1\1-6>a

E:\ms\AspectJ\1\1-6>ajc *.java -d .

E:\ms\AspectJ\1\1-6>b

E:\ms\AspectJ\1\1-6>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice attached to the execute pointcut
Captured this reference: com.aspectj.Testmain@3e25a5
Inside hh(int,String)
E:\ms\AspectJ\1\1-6>
```

**Figure 8. Program Running Result of Exp. 4**

The "Before()" advice can access the referenced object identifier, "this" points to the object during method execution. The way to do so is as follows: Include the "myobject" identifiers inside it, then, bind it to "TDExe (Testmain)" pointcut.

## 7. Capture of Object-based Join Points

This ([Type|Identifier]) refers to the object type referenced by this at the captured join point. The join point's target depends on its type [6], and its args is the available parameter at the captured join point. Static Type provides a judgment for the object type at the join point during static compiling and adopts the type of fully qualified class name. Identifier provides such a method that references an actual runtime object, through which to judge the actual type of the runtime object at the join point rather than the static type [7] only.

## 7.1. Capture of the Time When this References a Specific Type Problem

If you want to capture all join points, and the contents referenced by this at these join points are of a specific type, you can use this ([Type|Identifier]) pointcut [8]. This ([Type|Identifier]) pointcut is for checking the this reference at the captured join points, and determine whether to call the related notice. Its main characters [9] include: 1. when the running object is the designated type, it will capture all join points. 2. The define parameter of Type must be analyzed to a valid class. 3. Identifier is for checking the type of the runtime object referenced by this, and demonstrates the object to notice as required. 4. When using the * wildcard character, you are allowed to designate a valid object, and the reference at the join point must point to such object. 5. The join point emerges on the exception handling cannot use the value of the handler (TypePattern) pointcut.

The following Example 5 describes the application of the this ([Type|Identifier]) pointcut: 1. When the this reference is pointed to the any object of the Testmain type, the Identifier will be used for capturing the join point; 2. When the object referenced by this is of the AnotherTestmain type, Type will be used for capturing. Figure 9 demonstrates how to apply the this ([Type|Identifier]) pointcut.

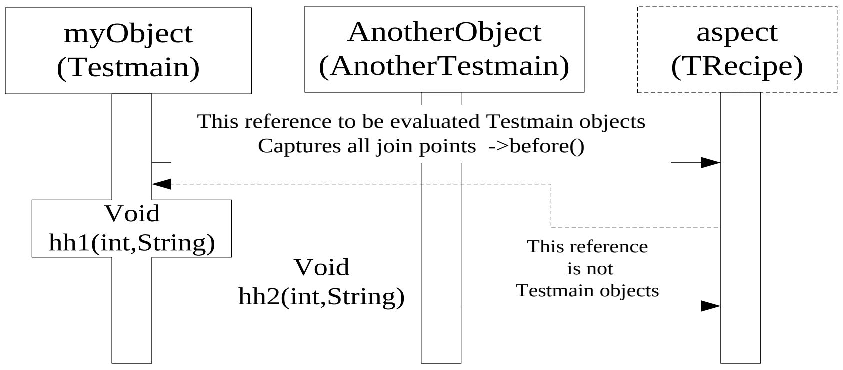


**Figure 9. How to Apply the this ([Type | Identifier]) Pointcut**

**Exp. 5: Use of the this ([Type|Identifier]) pointcut to capture the join points based on the this reference type.**

1) Business Logic Java Class

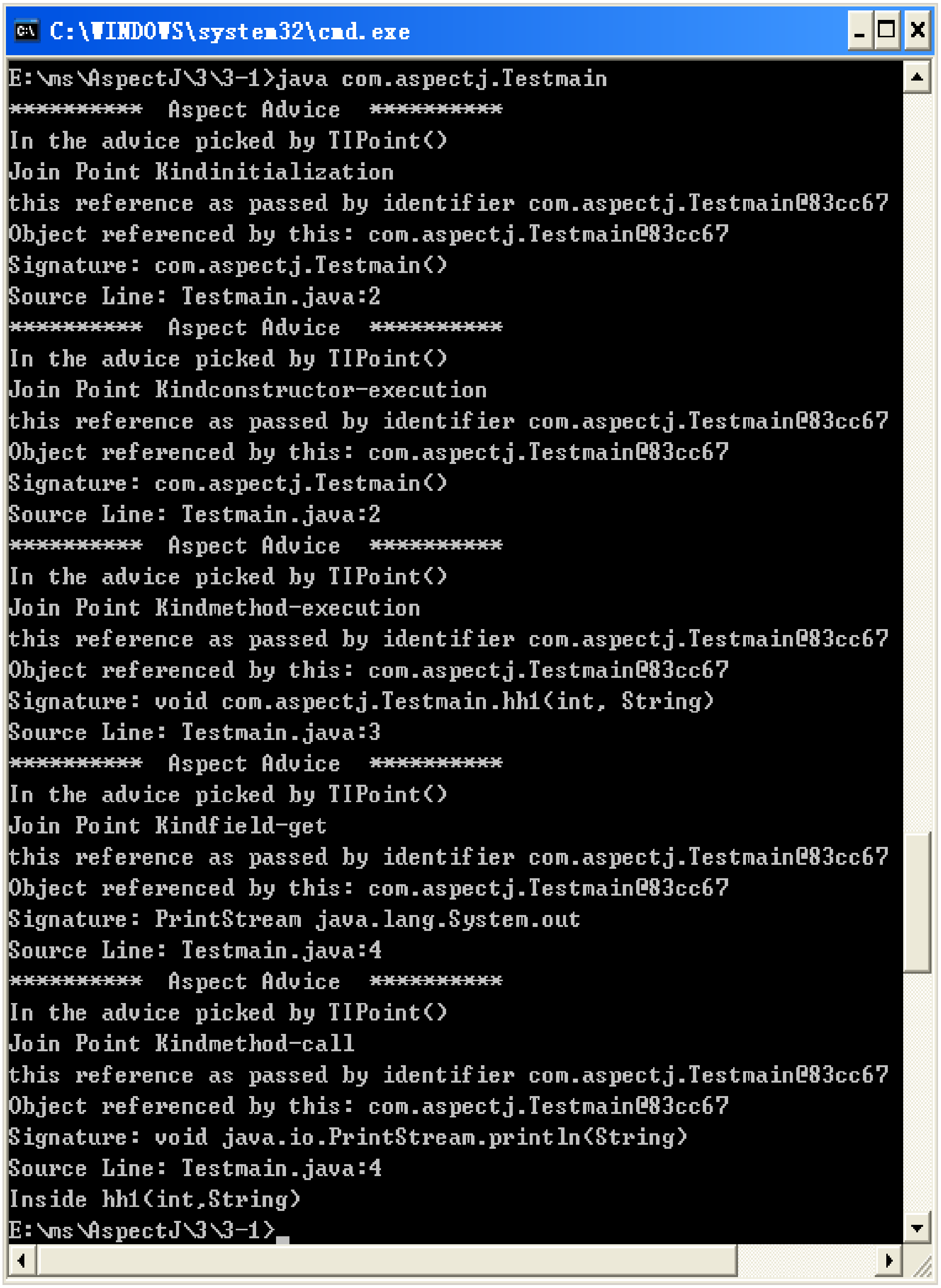
```
Testmain.java
package com.aspectj;
   public class Testmain{
      public void hh1(int number,String name){
         System.out.println("Inside hh1(int,String)");
      }
      public static void main(String[] args){
            //Create an instance of Testmain
           Testmain myObject = new Testmain ();
           //Make the call to hh1
           myObject.hh1(3, "M Zhang1");
```

```
            }
          }
        AnotherTestmain.java
         package com.aspectj;
          public class AnotherTestmain{
             public void hh2(int number,String name){
                  System.out.println("Inside hh2(int,String)");
             }
             public static void main(String[] args){
            AnotherTestmain myObject=new AnotherTestmain ();
             myObject.hh2(3, "M Zhang2");//Make the call to hh2
            }
         }
```

2) TRecipe Aspect of AspectJ

```
        package com.aspectj;
           public aspect TRecipe{
              pointcut TIPoint(Testmain object):this(object);
              //Type Pattern:AnotherTestmain
                pointcut TTPoint():this(AnotherTestmain);
               //Advice declaration
               before(Testmain object): TIPoint (object) && !within(TRecipe+){
                    System.out.println("**********  Aspect Advice  **********");
                   System.out.println("In the advice picked by TIPoint()");
                   System.out.println("Join Point Kind" + thisJoinPoint.getKind());
                   System.out.println("this reference as passed by identifier " + object);
                   System.out.println("Object referenced by this: " + thisJoinPoint.getThis());
                   System.out.println("Signature: " + thisJoinPoint.getStaticPart().getSignature());
                   System.out.println("Source                    Line:                 "                  +
     thisJoinPoint.getStaticPart().getSourceLocation());
               }
              before(): TTPoint () && !within(TRecipe+){
                   System.out.println("Join Point Kind" + thisJoinPoint.getKind());
                   System.out.println("Object referenced by this: " + thisJoinPoint.getThis());
              }
          }
```

After saving those three files under the same directory and running the ajc command for compiling the files, the.class files of the Aspect and Class will be created (that is,Testmain.class, AnotherTestmain.class and TRecipe.class).By running the java command, the following aspect results can be obtained, as shown in Figure 10:



```
C:\WINDOWS\system32\cmd.exe
E:\ms\AspectJ\3\3-1>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice picked by TIPoint()
Join Point Kindinitialization
this reference as passed by identifier com.aspectj.Testmain@83cc67
Object referenced by this: com.aspectj.Testmain@83cc67
Signature: com.aspectj.Testmain()
Source Line: Testmain.java:2
**********  Aspect Advice  **********
In the advice picked by TIPoint()
Join Point Kindconstructor-execution
this reference as passed by identifier com.aspectj.Testmain@83cc67
Object referenced by this: com.aspectj.Testmain@83cc67
Signature: com.aspectj.Testmain()
Source Line: Testmain.java:2
**********  Aspect Advice  **********
In the advice picked by TIPoint()
Join Point Kindmethod-execution
this reference as passed by identifier com.aspectj.Testmain@83cc67
Object referenced by this: com.aspectj.Testmain@83cc67
Signature: void com.aspectj.Testmain.hh1(int, String)
Source Line: Testmain.java:3
**********  Aspect Advice  **********
In the advice picked by TIPoint()
Join Point Kindfield-get
this reference as passed by identifier com.aspectj.Testmain@83cc67
Object referenced by this: com.aspectj.Testmain@83cc67
Signature: PrintStream java.lang.System.out
Source Line: Testmain.java:4
**********  Aspect Advice  **********
In the advice picked by TIPoint()
Join Point Kindmethod-call
this reference as passed by identifier com.aspectj.Testmain@83cc67
Object referenced by this: com.aspectj.Testmain@83cc67
Signature: void java.io.PrintStream.println(String)
Source Line: Testmain.java:4
Inside hh1(int,String)
E:\ms\AspectJ\3\3-1>_
```

**Figure 10. Running Result of Exp.5**

In the above TRecipe aspect, line 2 declares an aspect while line 3 and line 5 declare the named pointcut logic. The pointcut logic designates the join points of the application. In this example, it captures the type of the Testmain object referenced by this at the join point as well as all the join points of AnotherTestmain. The pointcut is named as TIPoint and TTPoint respectively. Lines 7 to 23 declare a single notice block. The before() notice simply points out that it will be run before any join point matched to the TIPoint pointcut. Lines 24 to 29 declare a single notice block. The before() notice simply points out that it will be run before any join point matched to the TTPoint pointcut.

## 7.2. Capture of the Time When the Target Object of the Join Point is a Specific Type

If you want to capture all join points when the target objects are of the designated type, you can consider using the target ([Type|Identifier]) pointcut. Its main characters [10] include: 1. when the target object is of the designated type, it will select all join points. 2. The define parameter of Type must be analyzed to a valid class, so as to select the related join point. 3. Identifier is for checking the type of the runtime object referenced at the captured join point, and demonstrates the object to notice as required. 4. When using the * wildcard character to point a join point, a target must be provided.

The following Example 6 describes the application of the target ([Type|Identifier]) pointcut: 1. When the target is the Testmain object pointed by Identifier, capture the join point; 2. When the target called by the method is of the AnotherTestmain type designated by Type, capture the join point. Figure 11 demonstrates how to apply the target ([Type|Identifier]) pointcut.

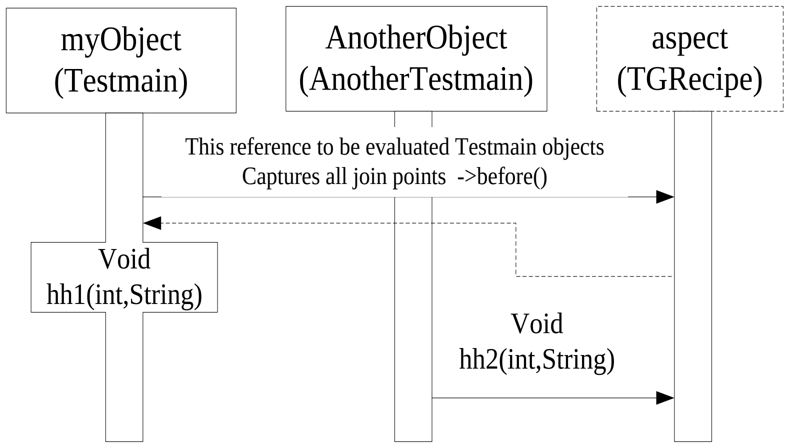
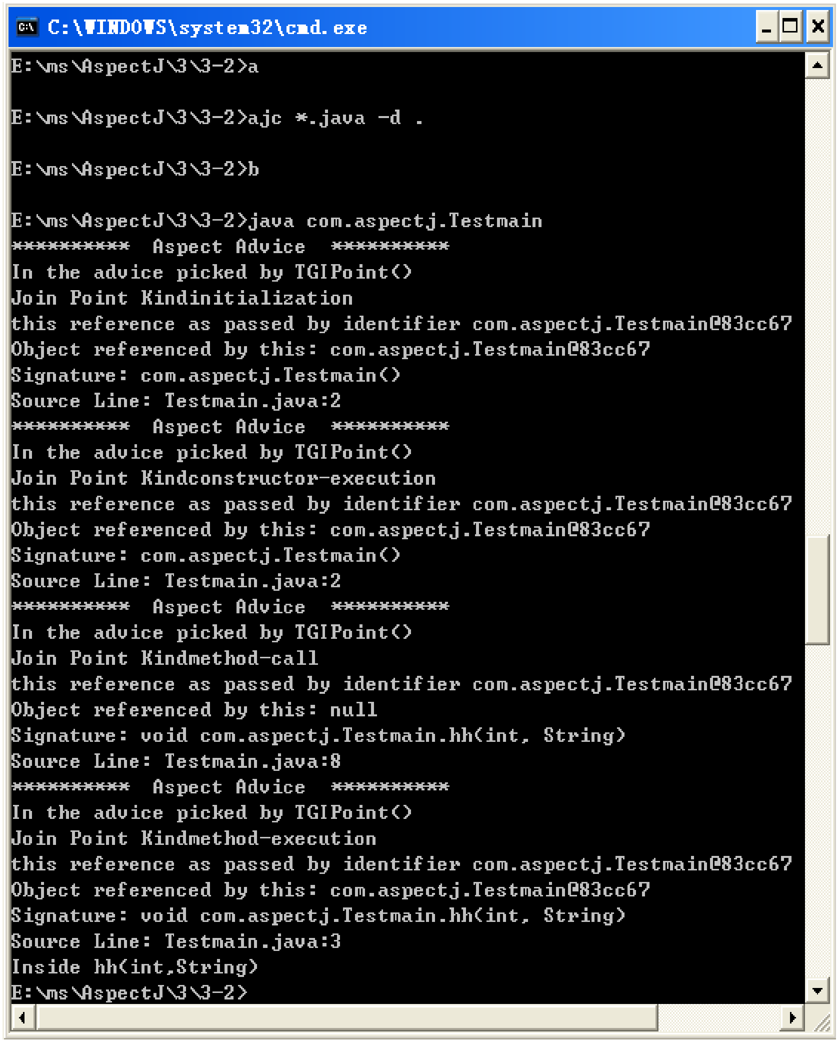

myObject (Testmain)
AnotherObject (AnotherTestmain)
aspect (TGRecipe)
This reference to be evaluated Testmain objects
Captures all join points ->before()
Void hh1(int,String)
Void hh2(int,String)

**Figure 11. How to Apply the Target ([Type|Identifier]) Pointcut**

**Exp. 6: Use of the target ([Type|Identifier]) pointcut to capture the join points based on the method target type.**

```
package com.aspectj;
  public aspect TGRecipe{
     pointcut TGIPoint(Testmain object):target(object);
     pointcut TTPPoint():target(AnotherTestmain);
     before(Testmain object): TGIPoint(object) && !within(TGRecipe+){
         System.out.println("**********  Aspect Advice  **********");
         System.out.println("In the advice picked by TGIPoint()");
         System.out.println("Join Point Kind" + thisJoinPoint.getKind());
         System.out.println("this reference as passed by identifier " + object);
         System.out.println("Object referenced by this: " + thisJoinPoint.getThis());
         System.out.println("Signature: " + thisJoinPoint.getStaticPart().getSignature());
         System.out.println("Source Line: " + thisJoinPoint.getStaticPart().getSourceLocation());
    }
    before(): TTPPoint () && !within(TGRecipe+){
         System.out.println("Join Point Kind" + thisJoinPoint.getKind());
         System.out.println("Signature: " + thisJoinPoint.getStaticPart().getSignature());
  }
 }
```

The aspect's running result is as shown in Figure 12:



```
C:\WINDOWS\system32\cmd.exe
E:\ms\AspectJ\3\3-2>a

E:\ms\AspectJ\3\3-2>ajc *.java -d .

E:\ms\AspectJ\3\3-2>b

E:\ms\AspectJ\3\3-2>java com.aspectj.Testmain
**********  Aspect Advice  **********
In the advice picked by TGIPoint()
Join Point Kindinitialization
this reference as passed by identifier com.aspectj.Testmain@83cc67
Object referenced by this: com.aspectj.Testmain@83cc67
Signature: com.aspectj.Testmain()
Source Line: Testmain.java:2
**********  Aspect Advice  **********
In the advice picked by TGIPoint()
Join Point Kindconstructor-execution
this reference as passed by identifier com.aspectj.Testmain@83cc67
Object referenced by this: com.aspectj.Testmain@83cc67
Signature: com.aspectj.Testmain()
Source Line: Testmain.java:2
**********  Aspect Advice  **********
In the advice picked by TGIPoint()
Join Point Kindmethod-call
this reference as passed by identifier com.aspectj.Testmain@83cc67
Object referenced by this: null
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:8
**********  Aspect Advice  **********
In the advice picked by TGIPoint()
Join Point Kindmethod-execution
this reference as passed by identifier com.aspectj.Testmain@83cc67
Object referenced by this: com.aspectj.Testmain@83cc67
Signature: void com.aspectj.Testmain.hh(int, String)
Source Line: Testmain.java:3
Inside hh(int,String)
E:\ms\AspectJ\3\3-2>
```

**Figure 12. Running Result of Exp.6**

## 8. Conclusion

By researching join points information captured in methods and Object by AspectJ, the capture of join-point information in program methods is achieved. AspectJ is the aspect-oriented extension of Java language, and aspects can be used in different stages of development and achieved in the case of no change of the original codes. AspectJ development is still at an early stage of exploration. Although AspectJ AOP realization with AspectJ has been more mature, the advantages and disadvantages co-exist and need to be further explored.

In this paper, the main ideas and contents are from “AspectJ Cookbook” by Russ Miles. On this basis, result of each sample and detailed description, are the major highlights of this paper in order to leave readers deeper impression. And, this is hereby the declaration.

## 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. de Oliveira Valente, “Object-oriented Transformations for Extracting Aspects”, Information and Software Technology, vol. 51, no. 1, **(2009)**, pp. 138-149.
[2] C. Jing, X. Bao-Wen, Z. Xiao-Yu, Q. Ju and Y. Bin, “Optimizing AspectJ Dynamic Advices Weaving Based on Aspect-Oriented Call Graph”, Journal of software, vol. 19, no. 9, **(2008)** September, pp. 2218-2227.
[3] Y. Yin and Z. Ji-De, “Analysis of the AspectJ Abnormal Type of Fault”, Computer Programming Skills & Maintenance,vol. 21, no. 1, **(2010)**.
[4] M. Su, L. Qing-Shan and C. Peng-Gang, “Basic Information Needed in Reversing-Engineer by AspectJ”, Com-puter Systems & Applications,vol. 20, no. 2, **(2011)**.
[5] W. Qi-Qing, Editor, “JAVA Programming Series”, Metallurgical Industry Press, Beijing, **(2002)**.
[6] Y. Nan-Sheng, Editor, “Java Programming”, Wuhan University, Wuhan, **(2006)**.
[7] R. Miles, “AspectJ Cookbook”, Edited Chang Xiao-Bo,Tsinghua University,Beijing, vol. 1, **(2006)**, pp. 55-65.
[8] R. Miles, “AspectJ Cookbook”, Edited Chang Xiao-Bo,Tsinghua University, Beijing, vol. 1, **(2006)**, pp. 79-89.
[9] Z. We-Bo, International Conference on Electrical Insulating Materials and Electrical Engineering, **(2012)** May 25-27, Shenyang,Liaoning, China.
[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.