CS 683 Emerging Technologies
Spring Semester, 2003
Doc 4 AspectJ Syntax 2
Contents

Reference

The AspectJ Programming Guide

Reading

The AspectJ Programming Guide

Section 1 Getting Started with AspectJ
Section 2 The AspectJ Language

AspectJ Syntax 2
Viewing Aspect Source Code

ajc -preprocess @Hello.lst

Will write out the Java source generated to implement the aspect

```
public class Hello {
   public Hello() {
      this(5);
      System.out.println("Hello()");
   }

   public Hello(int x) {
      System.out.println("Hello(" + x + ")");
   }

   public static void main(String[] args ) {
      new Hello();
   }
}

public aspect HelloAspect {
   before() : call( Hello.new(..) ) {
      System.out.println( "call-before");
   }

   before() : execution(Hello.new(..)) {
      System.out.println( "execution-before");
   }
}
```

Generated Hello.java

```
/*   Generated by AspectJ version 1.0.6 */
public class Hello {
  public Hello() {
    this(5);
    HelloAspect.aspectInstance.before1$ajc();
    ;
    System.out.println("Hello()");
  }
  public Hello(int x) {
    super();
    HelloAspect.aspectInstance.before1$ajc();
    System.out.println("Hello(" + x + ")");
  }
  public static void main(String[] args) {
    Hello.new$constructor_call();
  }

  private static Hello new$constructor_call() {
    HelloAspect.aspectInstance.before0$ajc();
    return new Hello();
  }

}
```

Generated HelloAspect

```
/*   Generated by AspectJ version 1.0.6 */
public class HelloAspect {
  public final void before0$ajc() {
    System.out.println("call-before");
  }

  public final void before1$ajc() {
    System.out.println("execution-before");
  }

  public HelloAspect() {
    super();
  }
  public static HelloAspect aspectInstance;
  public static HelloAspect aspectOf() {
    return HelloAspect.aspectInstance;
  }

  public static boolean hasAspect() {
    return HelloAspect.aspectInstance != null;
  }

  static {
    HelloAspect.aspectInstance = new HelloAspect();
  }
}
```

cflow

cflow(Pointcut)

Picks out join points in the Pointcut and code called by the Pointcut

cflowbelow(Pointcut)

Picks out join points in the code called by the Pointcut but not directly in the Pointcut

Example

```
public class Hello {
   public void c() {
      System.out.println("C");
   }

   public void b() {
      System.out.println("B");
      c();
   }

   public void a() {
      System.out.println( "A");
      b();
   }

   public static void main(String[] args ) {
   Hello test = new Hello();
   test.c();
   test.b();
   test.a();
   }
}
```

Same behavior when referring to b()

```
public aspect HelloAspect {
   before() : call( * b() ) && withincode( * a() ) {
      System.out.println( "call-within");
   }

  before() : call( * b() ) && cflow( call(* a()) ) {
      System.out.println( "call-cflow");
   }
}
```

Output

```
C
B
C
A
call-within
call-cflow
B
C
```

Different behavior when referring to c()

```
public aspect HelloAspect {
   before() : call( * c() ) && withincode( * a() ) {
      System.out.println( "call-within");
   }

  before() : call( * c() ) && cflow( call(* a()) ) {
      System.out.println( "call-cflow");
   }
}
```

Output

```
C
B
C
A
B
call-cflow
C
```

Recursive calls

An example to show how to pick off the start and tail of recursion

```
public class Hello {

   public void a(int repeats) {
      System.out.println( "A" + repeats);
      if (repeats > 0)
         a( repeats - 1);
   }

   public static void main(String[] args ) {
   Hello test = new Hello();
   test.a(4);
   }
}
```

The Aspect

```
public aspect HelloAspect {
   before() : call( * a(int) ) && withincode( * a(int) ) {
      System.out.println( "call-within");
   }

  before() : call( * a(int) ) && !cflowbelow( call(* a(int)) ) {
      System.out.println( "call-cflow");
   }
}
```

Output

```
call-cflow
A4
call-within
A3
call-within
A2
call-within
A1
call-within
A0
```

Some More on Matching

Boxing

```
public class Hello {
   public void a(int repeats) {
      System.out.println( "A" + repeats);
   }

   public static void main(String[] args ) {
   Hello test = new Hello();
   test.a(4);
   }
}


public aspect HelloAspect {
   before(Object boxed) : call( * a(int) ) && args(boxed) {
      System.out.println( "Match " + boxed);
   }
}
```

Output

```
Match 4
A4
```

Wildcards Again

* matches zero or more characters except for “.”

.. matches any sequence of characters that start & end with a ‘.’

target(java.util.*)

picks out all types in the java.util package but not inner types
Does not pick out java.util.Map.Entry

target(java.util..)

Still does not pick out java.util.Map.Entry

target(java.util..*)
Picks out all types in java.util package
Picks out all inner types like java.util. Map.Entry
Picks out java.util.logging.Handler

Subtype Patterns

+

- Matches all subtypes
- Comes after a type name pattern

call(Hello+.new())
call ((Hello+ && ! Hello).new())

```
public class Hello {
   public Hello() {
      System.out.println("Hello Contructor" );
   }
   }

public class Child extends Hello {
   public Child() {
      System.out.println("Child Contructor" );
   }
}
```

Examples

```
public aspect HelloAspect {
  before() : call( Hello.new() ) {
    System.out.println( "Match ");
  }
}
```

new Hello() Produces

Match
Hello Constructor

new Child() Produces

Hello Constructor
Child Constructor

```
public aspect HelloAspect {
  before() : call( Hello+.new() ) {
    System.out.println( "Match ");
  }
}
```

new Hello() Produces

Match
Hello Constructor

new Child() Produces

Match
Hello Constructor
Child Constructor

```
public aspect HelloAspect {
  before() : call( (Hello+ && ! Hello).new() ) {
    System.out.println( "Match ");
  }
}
```

new Hello() Produces

Hello Constructor

new Child() Produces

Match
Hello Constructor
Child Constructor

After Returning

after() returning()
Run after a join point returns normally

```
public class Hello {
   int increase(int value) {
      return value + 1;
   }

  public static void main(String[] args ) {
   Hello test = new Hello();
   test.increase(5);
  }
}

public aspect HelloAspect {
  after() returning() : call( int increase(int) ) {
    System.out.println( "No parameters ");
  }

  after() returning(Object returnBoxed) : call( int increase(int) ) {
    System.out.println( "Returned " + returnBoxed);
  }

  after(int arg) returning(int returned) : call( int increase(int) ) && args(arg) {
    System.out.println( "In " + arg + " Out " + returned);
  }
}
```

Output

```
No parameters
Returned 6
In 5 Out 6
```

Exceptions
Catching & After

```
import java.io.IOException;

public class Hello {

   int a(int value) throws IOException {
      if (value == 3)
         throw new IOException();
      System.out.println( "In a");
      return value + 1;
   }

   public static void main(String[] args ) {
      Hello test = new Hello();
      try {
         test.a( Integer.parseInt(args[0]) );
      }
      catch (IOException example) {
         System.out.println( "In Handler");
      }
   }
}
```

The Aspects

```
import java.io.IOException;

public aspect HelloAspect {
   after()  : call( int a(int) ) {
      System.out.println( "After ");
   }

   after()  returning : call( int a(int) ) {
      System.out.println( "returning ");
   }

   after()  throwing : call( int a(int) ) {
      System.out.println( "throwing ");
   }

   after() :handler(IOException) {
   System.out.println("After handler");
   }
}
```

Running java Hello 4

In a
After
Returning

Running java Hello 3

After
throwing
In Handler
After handler

Warning about not importing Exception

The following aspect does compile and will run, but not the way one thinks

```
public aspect HelloAspect {
   after()  : call( int a(int) ) {
      System.out.println( "After ");
   }

   after()  returning : call( int a(int) ) {
      System.out.println( "returning ");
   }

   after()  throwing : call( int a(int) ) {
      System.out.println( "throwing ");
   }

   after() :handler(IOException) {
   System.out.println("After handler");
   }
}
```

Running java Hello 3

After
throwing
In Handler

Throwing Checked Exceptions in Advice

Advice must explicitly declare if it throws an exception

```
import java.io.IOException;

public aspect HelloAspect {
   before() throws IOException : call( int a(int)) {
      throw new IOException();
   }
   after() throws IOException : call( int a(int)) {
      throw new IOException();
   }

   after() throwing(IOException e) throws IOException  : call( int a(int)) {
      throw new IOException();
   }
}
```

Restrictions on Throwing Exceptions

Exceptions a join point in AspectJ may throw are:

- method call and execution

  Checked exceptions declared by the target method's `throws` clause

- constructor call and execution

  Checked exceptions declared by the target constructor's `throws` clause

- field get and set

  No checked exceptions can be thrown from these join points

- exception handler execution

  Exceptions that can be thrown by the target exception handler

- static initializer execution

  no checked exceptions can be thrown from these join points

- pre-initialization, and initialization

  Any exception that is in the throws clause of *all* constructors of the initialized class

Illegal Example

This advice will not compile

```
public class Hello {
   void b() {
      System.out.println("In b");
   }
}

import java.io.IOException;

public aspect HelloAspect {
   before() throws IOException : call( void b()) {
      throw new IOException();
   }
}
```

Throwing Uncheck Exceptions in Advice

Unchecked exceptions can be thrown in advice

```
public class Hello {
   void b() {
      System.out.println("In b");
   }

   public static void main(String[] args ) {
      Hello test = new Hello();
      test.b();
   }
}

public aspect HelloAspect {
   before()  : call( void b()) {
      throw new ArithmeticException ();
   }
}
```

Warnings & Errors

Can specify that a join point should not be reached

Compiler will signal the problem

Forms:

declare error : Pointcut : String

declare warning: Pointcut : String

Example

```
public class Hello {
   void b() {
      System.out.println("In b");
   }

   public static void main(String[] args ) {
      Hello test = new Hello();
      test.b();
   }
}

public aspect HelloAspect {
   declare error  : call( void b()) : "Error string displayed by compiler";

}
```

Aspect Extension

An aspect may

- Extend an abstract aspect
- Extend a class
- Implement an interface

A class cannot extend an aspect

Aspect Extending an Aspect Example

```
public abstract  aspect ParentAspect {
   pointcut methodB() : call(* b(..) );
}

aspect ChildAspect extends ParentAspect {
   before() : methodB() {
      System.out.println(" Advise " );
   }
}
```

Aspect Extending a Class

```
public class Hello {
   void b() {
      System.out.println("In b");
   }
}

public  aspect HelloAspect extends Hello {
   before() : call(* foo() ) {
      b();
   }
}
```

Aspect State & Methods

Aspects can have state and methods

Aspects normally are created as singletons

AspectName.aspectOf() returns the singleton

```
public  aspect HelloAspect {

   private int callCount = 0;

   public int getCount() {
      return callCount;
   }

   public static void main(String[] arguments ) {
      HelloAspect aspect = HelloAspect.aspectOf();
      System.out.println( "Count is " + aspect.getCount() );
   }

   before() : call(* *(..) ) {
      callCount++;
   }
}
```

Result of running java HelloAspect

Count is 3