CS 683 Emerging Technologies
Spring Semester, 2003
Doc 5 AspectJ Syntax 3

Contents

Reference

The AspectJ Programming Guide

Reading

The AspectJ Programming Guide

Section 1 Getting Started with AspectJ
Section 2 The AspectJ Language

Non-Singleton Aspects

aspect perthis(Pointcut) { blah }

An instance of the aspect is created for every object that is the executing object of any join points of the Pointcut

aspect pertarget(Pointcut) { blah }

An instance of the aspect is created for every object that is the target object of any join points of the Pointcut

ajc would not compile examples with pertarget

aspect percflow(Pointcut) { blah }

An instance of the aspect is created for flow of control of the join points of the Pointcut

Created when flow in entered

aspect percflowbelow(Pointcut) { blah }

An instance of the aspect is created for flow of control of the join points of the Pointcut

Created when flow in entered

Non-Singleton Hello Class for Examples

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

percflow

```
public  aspect HelloAspect percflow(methodCall() ) {

   private int callCount = 0;
   private static int totalCount = 0;

   public int getCount() {
      return callCount;
   }

   public int getTotalCount() {
      return totalCount;
   }

   public static void main(String[] arguments ) {
      Hello example = new Hello();
      example.b();
      HelloAspect aspect = HelloAspect.aspectOf();
      System.out.println( "Object Count is " + aspect.getCount()
         + " Total count " + aspect.getTotalCount() );
   }

   pointcut methodCall() : call(* *(..) );

   before(): methodCall() {
      callCount++;
      totalCount++;
   }
```

Output of java HelloAspect

```
In b
Object Count is 1 Total count 6
```

perthis

```
public  aspect HelloAspect perthis(methodCall() ) {

   private int callCount = 0;
   private static int totalCount = 0;

   public int getCount() {
      return callCount;
   }

   public int getTotalCount() {
      return totalCount;
   }

   public static void main(String[] arguments ) {
      Hello example = new Hello();
      example.b();
      example.b();
      (new Hello()).b();
      HelloAspect aspect = HelloAspect.aspectOf(example);
      System.out.println( "Object Count is " + aspect.getCount()
         + " Total count " + aspect.getTotalCount() );
   }

   pointcut methodCall() : call(* *(..) );

   before(): methodCall() {
      callCount++;
      totalCount++;
   }
}
```

java HelloAspect

```
In b
In b
In b
Object Count is 2 Total count 3
```

Privileged Aspects

Privileged aspects have access to all members of other classes & aspects

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

public privileged  aspect HelloAspect {
   public static void main(String[] arguments ) {
      Hello example = new Hello();
      example.b();
   }
}
```

How this Works

Code generated by ajc for Hello.java

```
/*   Generated by AspectJ version 1.0.6 */

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

  public Hello() {
    super();
  }

  public final void b$ajc$backdoor() {
    this.b();
  }
}
```

Information about Join Points

thisJoinPoint and thisJoinPointStaticPart
  Provide information about the join point selected by a pointcut

Download the AspectJ documentation and view the JavaDoc for the org.aspectj.lang classes

Example

```
public class Hello {
   int b(int foo, String bar) {
      System.out.println("In b");
      return foo + 1;
   }
}
```

Aspect

```
import org.aspectj.lang.*;
import org.aspectj.lang.reflect.CodeSignature;

public privileged  aspect HelloAspect {
   public static void main(String[] arguments ) {
      Hello example = new Hello();
      int answer = example.b(5, "cat");
   }

   Object around() : call(* b(..) ) {
      System.out.println("Intercepted message: " +
         thisJoinPointStaticPart.getSignature().getName());
      System.out.println("In class: " +
         thisJoinPointStaticPart.getSignature().getDeclaringType().getName());
      printParameters(thisJoinPoint);
      Object result = proceed();
      System.out.println( "Result " + result);
      return result;
   }

   private void printParameters(JoinPoint jp) {
      System.out.println("Arguments: " );
      Object[] args = jp.getArgs();
      String[] names =
((CodeSignature)jp.getSignature()).getParameterNames();
      Class[] types = ((CodeSignature)jp.getSignature()).getParameterTypes();
      for (int i = 0; i < args.length; i++) {
         System.out.println(" " + i + ". " + names[i] +
            " : " + types[i].getName() +
            " = " + args[i]);
      }
   }
 }
```

Result of running java HelloAspect

```
Intercepted message: b
In class: Hello
Arguments:
 0. foo : int = 5
 1. bar : java.lang.String = cat
In b
Result 6
```

Order & Conflicts Among Aspects

Multiple pieces of advice may apply to the same join point

Advice precedence determines the order to apply the advice

Determining precedence
Advice in Different Aspects

- If aspect A dominates aspect B, then advice in A has precedence over advice in B.

```
aspect B { blah
}

aspect A dominates B { blah
}
```

- If aspect A is a subaspect of aspect B, then advice in A has precedence over advice in B.

```
abstract aspect B {
}

public aspect A extends B {
}
```

- Unless otherwise specified with the dominates keyword, advice in a subaspect dominates advice in a superaspect.

- Otherwise, if two pieces of advice are defined in two different aspects, it is undefined which one has precedence.

Determining precedence
Advice in the same Aspect

- If either are after advice, then the one that appears later in the aspect has precedence over the one that appears earlier.
- Otherwise, then the one that appears earlier in the aspect has precedence over the one that appears later.

Circularities

```
aspect A {
  before(): execution(void main(String[] args)) {}
  after(): execution(void main(String[] args)) {}
  before(): execution(void main(String[] args)) {}
}
```

Circularities in using the rules are compile errors

Effects of precedence

At a particular join point, advice is ordered by precedence

around advice

If calls proceed then advice with next precedence is run

Otherwise advice with lower precedence at this join point is not run

before advice

If throws an exception advice with lower precedence is not run