JBoss AOP

para quem já conhece AspectJ

Francisco Reverbel
Departmento de Ciência da Computação
Universidade de São Paulo

Características do JBoss AOP

- Arcabouço para programação OA dinâmica
- Suporte a programação orientada a aspectos em Java 100% puro
  - Não é uma extensão à linguagem Java
- Software livre (LGPL)
- Produzido pelo mesmo grupo que criou o servidor de aplicações JBoss
  - Independente do servidor JBoss
  - Pode ser usado dentro e fora do servidor JBoss

Características do JBoss AOP

- Aspectos são classes Java normais
- Conselhos são métodos normais
  - definidos numa classe aspecto
  - assinatura de um conselho:

Aqui poderia aparecer uma das implementações da interface `Invocation` definidas no pacote `org.jboss.aop.joinpoint`

```
import org.jboss.aop.joinpoint.Invocation;

Object someAdvice(Invocation inv) throws Throwable;
```

- Duas alternativas para definição de cortes pontuais e para associação entre cortes pontuais e conselhos
  - XML
  - Anotações de Java 5.0

Um aspecto simples

```
import org.jboss.aop.joinpoint.Invocation;

public class TracingAspect {

  public Object trace(Invocation inv)
         throws Throwable {
    System.out.println("Entering");
    try {
      return inv.invokeNext();
    }
    finally {
      System.out.println("Leaving");
    }
  }

}
```

O aspecto é uma classe normal

Assinatura de um conselho

Isto roda "antes"

Isto é o "proceed"

Isto roda "depois"

Ligação de cortes pontuais a conselhos em XML

```
<aop>

  <aspect class="TracingAspect" />

  <bind pointcut="execution(void SomeClass->someMethod())">
    <advice name="trace" aspect="TracingAspect" />
  </bind>

  <bind pointcut="execution(* OtherClass->*(int))">
    <advice name="trace" aspect="TracingAspect" />
  </bind>

  <bind pointcut="execution(double com.acme.*->*(double))">
    <advice name="trace" aspect="TracingAspect" />
  </bind>

</aop>
```


Ligação de cortes pontuais a conselhos com anotações Java

```
import org.jboss.aop.Aspect;
import org.jboss.aop.Bind;
import org.jboss.aop.joinpoint.Invocation;

@Aspect public class TracingAspect {

  @Bind(pointcut="execution(double com.acme.*->*(double))")
  public Object trace(Invocation inv)throws Throwable {
    System.out.println("Entering");
    try {
      return inv.invokeNext();
    }
    finally {
      System.out.println("Leaving");
    }
  }

}
```

Pontos de Junção

e correspondentes cortes pontuais primitivos

- **Execução de método**

```
execution(void com.acme.Foo->someMethod(int))
```

- **Execução de construtor**

```
execution(void com.acme.Foo->new(int, String))
```

- **Chamada a método**

```
call(void com.acme.Foo->someMethod(int))
```

- **Chamada a construtor**

```
call(void com.acme.Foo->new(int, String))
```

- **Leitura de campo**

```
get(int com.acme.Foo->someField)
```

- **Escrita em campo**

```
set(int com.acme.Foo->someField)
```


Pontos de Junção

em tempo de execução

- Instâncias de classes do pacote **`org.jboss.aop.joinpoint`** que implementam a interface **`Invocation`**:
  - **`InvocationBase`** (classe abstrata)
    - **`MethodInvocation`**
    - **`ConstructorInvocation`**
    - **`FieldInvocation`** (classe abstrata)
      - **`FieldReadInvocation`**
      - **`FieldWriteInvocation`**
    - **`MethodCalledByMethodInvocation`**
    - **`MethodCalledByConstructorInvocation`**
    - **`ConstructorCalledByMethodInvocation`**
    - **`ConstructorCalledByConstructorInvocation`**

Sobrecarga de conselhos

```
public class TracingAspect {

  public Object trace(MethodInvocation inv) throws Throwable {
    System.out.println("Entering method " + inv.getMethod());
    try {
      return inv.invokeNext();
    }
    finally {
      System.out.println("Leaving method " + inv.getMethod());
    }
  }
  public Object trace(ConstructorInvocation inv) throws Throwable {
    System.out.println("Entering constructor " + inv.getConstructor());
    try {
      return inv.invokeNext();
    }
    finally {
      System.out.println("Leaving constructor " + inv.getConstructor());
    }
  }
}
```


XML para os conselhos sobrecarregados

```
<aop>

  <aspect class="TracingAspect" />

  <bind pointcut="execution(public * com.acme.SomeClass->*(..))">
    <advice name="trace" aspect="TracingAspect" />
  </bind>

  <bind pointcut="execution(public com.acme.SomeClass->new(..))">
    <advice name="trace" aspect="TracingAspect" />
  </bind>

</aop>
```

Anotações nos conselhos sobrecarregados

```
@Aspect public class TracingAspect {

  @Bind(pointcut="execution(public * com.acme.SomeClass->*(..))")
  public Object trace(MethodInvocation inv) throws Throwable {
    System.out.println("Entering method " + inv.getMethod());
    try {
      return inv.invokeNext();
    } finally {
      System.out.println("Leaving method " + inv.getMethod());
    }
  }
  @Bind (pointcut="execution(public com.acme.SomeClass->new(..))")
  public Object trace(ConstructorInvocation inv) throws Throwable {
    System.out.println("Entering constr. " + inv.getConstructor());
    try {
      return inv.invokeNext();
    } finally {
      System.out.println("Leaving constr. " + inv.getConstructor());
    }
  }
}
```


Conselhos "em volta"

```
public class ExampleAspect {

  Object aroundAdvice(MethodInvocation inv)
          throws Throwable {

    ... // code that runs before the original method runs
    try {
      return inv.invokeNext();
    }
    finally {
      ... // code that runs after the original method runs
    }
  }

}
```

Conselhos "antes" e "depois"

```
public class ExampleAspect {

  Object beforeAdvice(MethodInvocation inv)
          throws Throwable {
    ... // code that runs before the original method runs
    return inv.invokeNext();
  }

  Object afterReturningAdvice(MethodInvocation inv)
          throws Throwable {
    Object obj = inv.invokeNext();
    ... // code that runs after the original method returns
    return obj;
  }

}
```


Conselhos "antes" e "depois"

```
public class ExampleAspect {

  Object afterThrowingAdvice(MethodInvocation inv)
          throws Throwable {
    try {
      return inv.invokeNext();
    }
    catch (Throwable t) {
      ... // code that runs after a throwable is thrown
      throw t;
    }
  }

}
```

Coringas em cortes pontuais

"*" é coringa "normal"
".." especifica número qualquer de parâmetros em métodos ou em construtores

```
call(void com.acme.Foo->someMethod(int))
call(public * com.acme.Foo->*(*))      métodos públicos de Foo com 1 param.
call(public * com.acme.Foo->*(..))     métodos públicos de Foo
call(public * com.acme.*->*(..))       métodos públicos em com.acme.*
call(public * *->*(..))                qualquer método público

call(void Person->setName(java.lang.String))
call(void Person->setPhone(*))
call(void Person->set*(*))
call(void *->set*(*))                  qualquer "setador"

call(Person->new(..))                  qualquer construtor de Person
call(*->new(..))                       qualquer construtor
```


Outros cortes pontuais

- **Qualquer ponto de junção na classe especificada**

```
all(com.acme.Foo)
all(com.acme.*)
```

- **Qualquer chamada a método/construtor feita de dentro da classe especificada**

```
within(com.acme.Foo)
within(com.acme.*)
```

- **Ou do método ou construtor especificado**

```
withincode(void com.acme.Foo->someMethod(int))
withincode(public com.acme.Foo->new())
```

- **Classe que tem o método ou construtor especificado**

```
has(void *->setName(String))
has(public *->new())
```

- **Classe que tem o campo especificado**

```
hasfield(int *->versionNumber)
hasfield(public java.lang.String *->*)
```

Interceptador:
aspecto com um só conselho

```
import org.jboss.aop.Interceptor;
import org.jboss.aop.joinpoint.Invocation;

public class TracingInterceptor
        implements Interceptor {

  public Object invoke(Invocation inv)
          throws Throwable {
    System.out.println("Entering");
    try {
      return inv.invokeNext();
    }
    finally {
      System.out.println("Leaving");
    }
  }

}
```

O interceptador implementa esta interface, que tem só o método invoke

Ele não precisa ser declarado como aspect no XML nem numa anotação

Ligação de cortes pontuais a interceptadores em XML

```
<aop>

  <!-- no aspect element for the interceptor -->

  <bind pointcut="execution(void SomeClass->someMethod())">
    <interceptor class="TracingInterceptor" />
  </bind>

  <bind pointcut="call(* OtherClass->*(int))">
    <interceptor class="TracingInterceptor" />
  </bind>

  <bind pointcut="call(double com.acme.*->*(double))">
    <interceptor class="TracingInterceptor" />
  </bind>

</aop>
```

Ligação de cortes pontuais a interceptadores com anotações

```
import org.jboss.Interceptor;
import org.jboss.aop.Bind;
import org.jboss.aop.joinpoint.Invocation;

public class TracingInterceptor implements Interceptor {

  @Bind(pointcut="call(double com.acme.*->*(double))")
  public Object invoke(Invocation inv)throws Throwable {
    System.out.println("Entering");
    try {
      return inv.invokeNext();
    }
    finally {
      System.out.println("Leaving");
    }
  }

}
```


Introduções

- Esta classe não é seriável:

```
public class POJO {
  private String field;
}
```

- Agora ela é:

```
<aop>

  <introduction class="POJO">
    <interfaces>java.io.Serializable</interfaces>
  </introduction>

</aop>
```

Mixins

- E se a interface introduzida tiver métodos?
  - Exemplo: java.io.Externalizable

```
<aop>

  <introduction class="POJO">
    <mixin>
      <interfaces>java.io.Externalizable</interfaces>
      <class>ExternalizableMixin</class>
      <construction>
         new ExternalizableMixin(this)
      </construction>
    </mixin>
  </introduction>

</aop>
```

  - O JBoss AOP põe na classe POJO um novo campo, com uma instância da classe mixin

Você escreve a classe mixin...

```
public class ExternalizableMixin
                implements java.io.Externalizable {
  POJO pojo;

  public ExternalizableMixin(POJO pojo) {
    this.pojo = pojo;
  }
  public void readExternal(java.io.ObjectInput in)
      throws IOException, ClassNotFoundException {
    pojo.field = in.readUTF();
  }

  public void writeExternal(java.io.ObjectOutput out)
      throws IOException {
    out.writeUTF(pojo.field);
  }

}
```