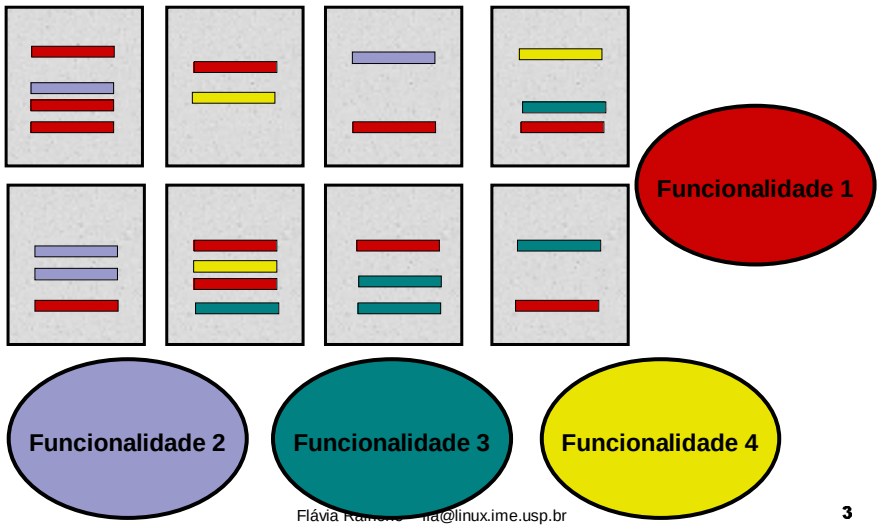
Programação Orientada a Aspectos

Bibliotecas Dinâmicas

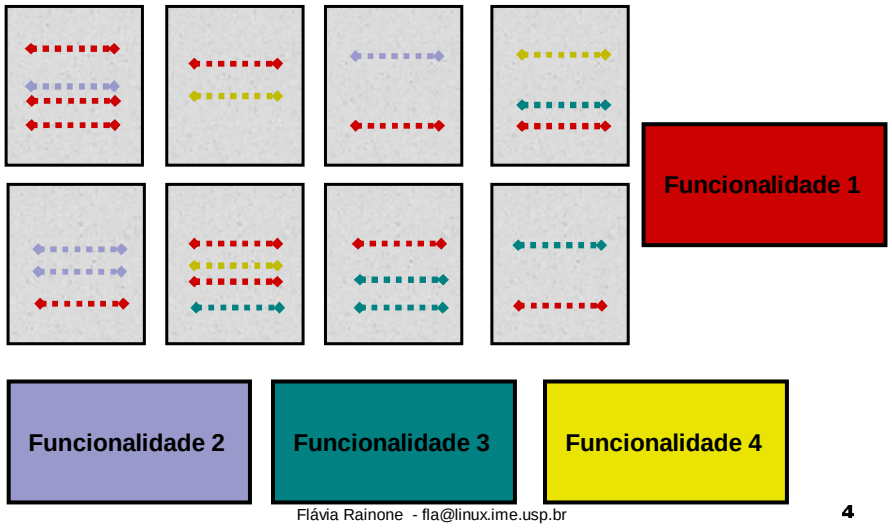
Motivação

```
class MyClass  {
       public MyClass() {
              ..........
              ..........
              ..........
       }
       public void method1 () {
              ..........
              ..........
       }
       public int method2() {
              ..........
              ..........
       }
       public boolean method3() {
              ..........
              ..........
       }
}
```

Motivação

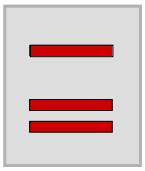


Motivação

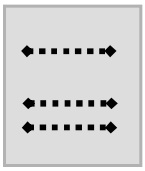


Conceitos

- Advice
- Introductions



- Pointcuts
- JoinPoints



- Aspects

Advice X Introduction

- ADVICE:

```
Class MyClass {
   void method1() {
       ..................
       ..................
       ..................
       ..................
       ..................
       ..................
       ..................
       ..................
       ..................
   }
}
```

- INTRODUCTION

```
class MyClass implements X {
   void method1() {
       ..................
   }
   void method2() {
      ..................
   }
   void methodXXX() {
      ..................
      ..................
      ..................
      ..................
    }
}
```

JBOSS AOP

JBOSS ASPECT ORIENTED PROGRAMMING FRAMEWORK

JBoss AOP

- Interceptor (advice), Introduction e Pointcuts
- Manipulação de ByteCodes + HotDeploy
- Uso de XML para conectar interceptors e introductions a classes java, através de definições de pointcuts. (META-INF/jboss-aop.xml)
- Expressões regulares para especificar pointcuts

Interceptor

```
public interface Interceptor {
  public String getName();
  public InvocationResponse
  invoke(Invocation invocation)
  throws Throwable;
}
InvocationResponse
  Invocation.invokeNext()
```


InterceptorFactory

```
import org.jboss.aop.Advisor;

public interface
  InterceptorFactory {
  public Interceptor
  create(Advisor advisor);
}
```

Pointcuts

- Método

```
<aop>
<interceptor name="Logging"
  factory="org.jboss.LoggingInterceptorFacto
  ry"/>
<method-pointcut class="com.acme.POJO"
  methodName="get.*">
  <interceptors>
     <interceptor-ref name="Logging"/>
  </interceptors>
</method-pointcut>
</aop>
```


Pointcuts

- Atributo

```
<aop>
<field-pointcut class="com.acme.POJO"
  fieldName=".*">
  <interceptors>
     <interceptor-ref name="Logging"/>
  </interceptors>
</field-pointcut>
</aop>
```

Pointcuts

- Construtores

```
<aop>
<constructor-pointcut class="com.acme.POJO">
  <interceptors>
     <interceptor-ref name="Logging"/>
  </interceptors>
</constructor-pointcut>
</aop>
```


Pointcuts

- Classe (Métodos, Construtores e Campos)

```
<aop>
<interceptor-pointcut
  class="com.acme.POJO.*">
  <interceptors>
     <interceptor
  factory="jboss.security.SecurityFactory"/>
  <interceptor-ref name="Logging"/>
  </interceptors>
</interceptor-pointcut>
</aop>
```

Pointcuts

- Caller

```
<aop>
<caller-pointcut
  class="com.acme.CallingPOJO"
  withinMethodName="callSomeMethod"
  calledClass="com.acme.POJO"
  calledMethod="someMethod">
  <interceptors>
     <interceptor-ref name="Logging"/>
  </interceptors>
</caller-pointcut>
</aop>
```


Introduction

-
```
<aop>
<introduction-pointcut
class="org.jboss.POJO">
<interfaces>
javax.jdo.spi.PersistenceCapable,
org.jboss.jdo.JBossJDO </interfaces>
</introduction-pointcut>
</aop>
```

-
```
POJO pojo = new POJO();
PersistenceCapable jdo =
(PersistenceCapable)pojo;
JBossJDO jboss_jdo = (JBossJDO)pojo;
```

Introduction

```
<aop>
<introduction-pointcut class="org.jboss.POJO">
  <mixin>
     <interfaces>
          javax.jdo.spi.PersistenceCapable,
          org.jboss.jdo.JBossJDO
     </interfaces>
     <class>org.jboss.jdo.JdoMixin</class>
     <construction>new
     org.jboss.jdo.JdoMixin(this, "param", 3)
     </construction>
  </mixin> </introduction-pointcut> </aop>
```


Meta Dados

- Uma forma de o interceptador extrair dados sobre uma classe para fazer o seu trabalho.
- Os meta dados podem ser fornecidos no xml, ou no comentário da classe, através de XDoclet

Definindo Meta Dados

```
<aop>
<interceptor name="TX"
  factory="org.jboss.aop.plugins.TxIntercept
  orFactory"/>
  <interceptor-pointcut
  class="com.acme.POJO.*">
     <interceptors>
          <interceptor-ref name="TX"/>
     </interceptors>
  </interceptor-pointcut>
```


Definindo Meta Dados

```
<class-metadata group="transaction"
  class="com.acme.POJO.*">
  <default>
     <trans-attribute>Required</trans-attribute>
  </default>
  <method name="someMethod.*">
  <trans-attribute>RequiresNew</trans-attribute>
  </method>
  <field name="stuff">
  <trans-attribute>Never</trans-attribute>
  </field>
</class-metadata>
```

Obtendo Meta Dados

- **Invocation.getMetaData(String group, String attribute)**
- **public class TxInterceptor implements Interceptor {**

```
public InvocationResponse invoke(Invocation invocation) throws Throwable {
   String txtype = (String)invocation.getMetaData("transaction", "trans-attribute");
}
}
```


Importando XML

- **<aop>**

```
<interceptor name="Logging" class="org.jboss.LoggingInterceptor" singleton="true">
<logging-level>verbose<logging-level>
</interceptor>
</aop>
```

- **public interface XmlLoadable {**

```
public void importXml(Element element) throws Exception;
}
```

Limitações

- Uma classe não será instrumentada se não tiver interceptors, poincuts, introductions ou class-matadata definidos antes de ser carregada, e não será possível fazer hot-deploy de quaisquer pointcuts AOP, ou invocar a interface de tempo de execução do JBossAOP de uma classe instrumentada. Caso queira fazer isso:

```
<aop> <advisable class=
  “com.pacote.Classe”/></aop>
```


Finalizando...

- É possível definir se métodos serão ou não instrumentados, construtores, e etc. Afinal, a instrumentação (inclusive de atributos) pode causar algum impacto na performance.
- Alguns métodos e campos são ignorados na definição de pointcuts.

Finalizando...

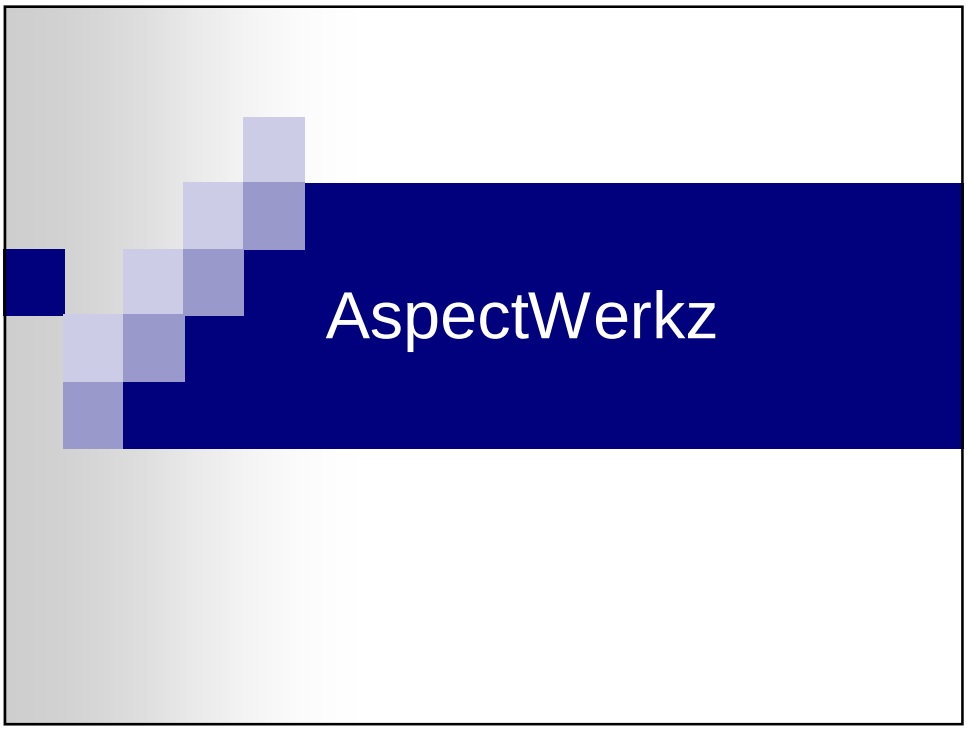
- É possível utilizar XDoclet para definir todos os elementos AOP do JBoss, como Interceptors, Pointcuts, Introduction, mas o suporte a metadados é provavelmente o mais útil.
-
```
public interface Advised {
  public Advisor _getAdvisor();
  public InstanceAdvisor
    _getInstanceAdvisor();
  }
```


Finalizando...

- JBoss AOP pode ser utilizado separadamente do JBoss, mas:
  - É preciso que JBossAOP tenha controle do ClassLoader, portanto é necessário utilizar o SystemClassLoader do JBoss AOP

```
java -Djava.system.class.loader =
  org.jboss.aop.standalone.SystemClassLoader
  POJO
```

- No JBoss, extensão .aop (para o jar)



AspectWerkz

- Advices
- Introductions
- Pointcuts
- Aspects
- Joinpoints

Advices

- AroundAdvice: é utilizado para interceptar invocações de métodos.
- PreAdvice: é executado antes do joinpoint.
- PostAdvice: é executado após o joinpoint.
- ThrowsAdvice: executado nos pontos em que uma exceção é lançada de um método.

Advices

```
public class MyAroundAdvice extends AroundAdvice
  {
  public MyAroundAdvice() {
     super();
  }
  public Object execute(final JoinPoint
  joinPoint) throws Throwable {
     // do some stuff
     Object result = joinPoint.proceed();
     // do some more stuff
     return result;
  }
}
```

Pointcuts

- MethodPointcut
  - AroundAdvice
- FieldPointcut
  - PreAdvice
  - PostAdvice
- ThrowsPointcut
  - ThrowsAdvice

Pointcuts

- CallerSidePointcut
  - PreAdvice
  - PostAdvice
- CFlowPointcut
  - deve ser utilizado juntamente com os outros pointcuts

Introductions

- Suporte a *mixin classes*.
- Muito similar às Introductions do JBossAOP.
- Caso tenha colisão de nome de métodos em uma ou mais Introductions, o resultado é indeterminado.

Aspects

- Em AspectWerkz, temos a definição de Aspects.
- O conceito é o mesmo que foi apresentado no início:
  - Unidade de modularidade que representa uma funcionalidade ortogonal à aplicação.
  - Composto por pointcuts, advices e introductions.

Exemplos

- **<advice-def name="advices/caching" class="advices.CachingAdvice" deployment-model="perInstance">**
  <param name="timeout" value="10"/>
  </advice>
- **<introduction-def name="java/io/Serializable" interface="java.io.Serializable"/>**
- **<introduction-def name="mixins/Mixin" interface="mixins.Mixin" implementation="mixins.MixinImpl" deployment-model="perThread"/>**

Exemplos

```
<aspect name="MyAspect">
  <pointcut-def name="facadeCalls"
  type="cflow" pattern="*
  *..facade.*.*(..)"/>
  <pointcut-def name="setters" type="method"
  pattern="String domain.*.set*(..)"/>
  <pointcut-def name="getters" type="method"
  pattern="String domain.*.get*(..)"/>
  <pointcut-def name="persistentFields"
  type="setField" pattern="* domain.*.*"
```

Exemplos

```
<advice cflow="facadeCalls" pointcut="setters
AND !getters">
   <advices-ref name="log_and_cache"/>
</advice>
<advice pointcut="persistentFields">
   <advice-ref name="persistent"/>
</advice>
<introduction class="domain.*">
   <introduction-ref name="serializable"/>
   <introduction-ref name="mixin"/>
</introduction>
</aspect>
```


Finalizando...

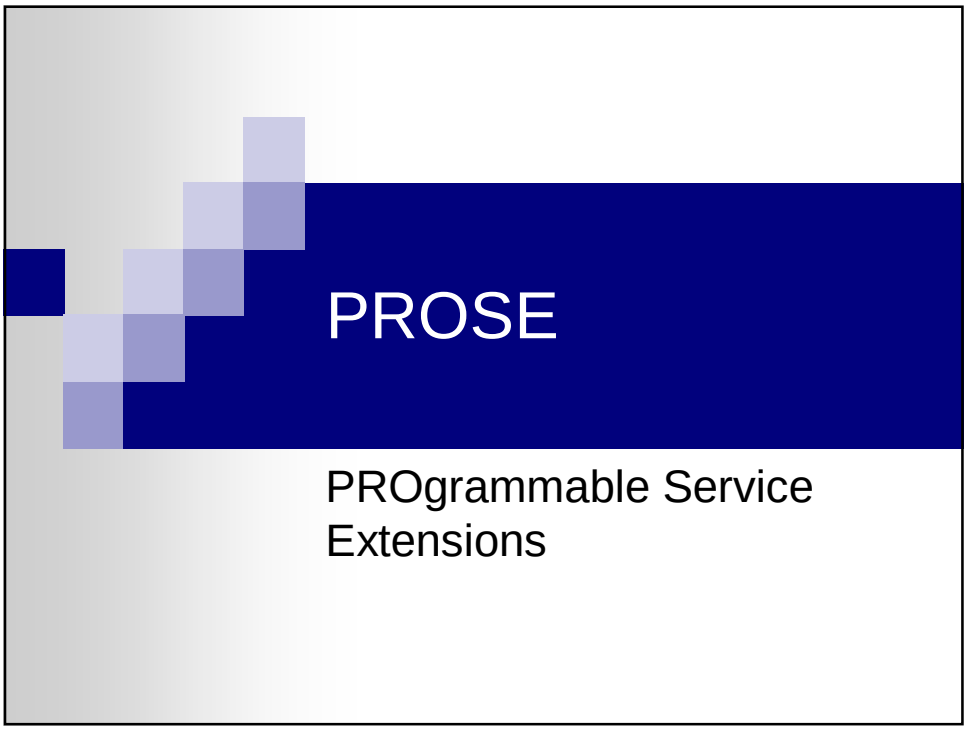
- Não utiliza expressões regulares, e sim expressões mais poderosas
  - Ex: org.POJO+, indica POJO e suas subclasses
- É possível especificar o pacote de um aspecto, fazer heranças de aspectos e até declarar aspectos abstratos no xml.
- Suporte a Metadados.

Finalizando

- Para reconfigurar e fazer hot deploy dos aspectos, é preciso utilizar código.
  - `AspectWerkz.getSystem(uuid).getIntroduction("mixins/Mixin").swapImplementation("mixins.NewMixinImpl");`
- Extensível: AdviceContainer e IntroductionContainer.

Finalizando...

- Esse se conecta abaixo do class loader bootstrap e pode alterar bytecodes de classes carregadas por todos os class loaders abaixo dele.
- Algumas forma que o AspectWerkz utiliza para fazer isso:
  - HotSwap (JVM 1.4)
  - Transparent BootClasspath (JVM 1.3)



PROSE

- Simples
- Poincuts
- Advice
- Aspect
- Baseado em AspectJ

Crosscut

- Pointcut + Advice
- Um pointcut é um objeto.
- Um advice é um método.
- Um pointcut é associado a um advice através de um crosscut, que também é um objeto.

Crosscut

- Exemplo:

```
public class ExampleCrosscut extends MethodCut {
   protected PointCutter pointCutter() {
     return (Executions.before().AND
     (Within.method("bar.*")) );
   }
   public void METHOD_ARGS(Foo x, String arg1,REST
     y) {
     System.err.println(" -advice: before "+ x +
     ".'bar*'("+ arg1 +",..) called");
   }
}
```

Crosscut

- SetCut
  - SET_ARGS((<TargetType> obj, <FieldType> f)
- GetCut
  - GET_ARGS(<TargetType> obj, <FieldType> f)
- ThrowCut
  - THROW_ARGS(<ExceptionType> e)

Aspect

- É um objeto, composto por um ou mais Crosscuts.
- Deve extender:
  - Classe abstrata Aspect:

```
protected abstract Crosscut[]
crosscuts();
```

  - Classe DefaultAspect.

Aspect

- Exemplo:

```
public class ExampleAspect extends Aspect {
  ExampleCrosscut myCrosscut = new
  ExampleCrosscut();

  public Crosscut[] crosscuts() {
     return new Crosscut[] {myCrosscut};
  }
}
```


Aspect

```
ExampleAspect asp = new ExampleAspect();
ProseSystem.getAspectManager().insert(asp);
```

Finalizando...

- Não possui around advice.
- Mas é extensível, você mesmo pode definir novos joinpoints.
- É possível adicionar e remover aspectos de uma máquina virtual em runtime.
  - Linha de comando.
  - Interface gráfica.

Conclusão

Conclusão

- JBossAOP:
  - Complexo.
  - Ferramenta poderosa.
  - Não suporta o conceito de Aspecto como unidade de encapsulamento.
  - Desenvolvida inicialmente voltada para o suporte de Middleware do JBoss.
  - Ainda não está finalizada, inclusive na sua versão standalone.

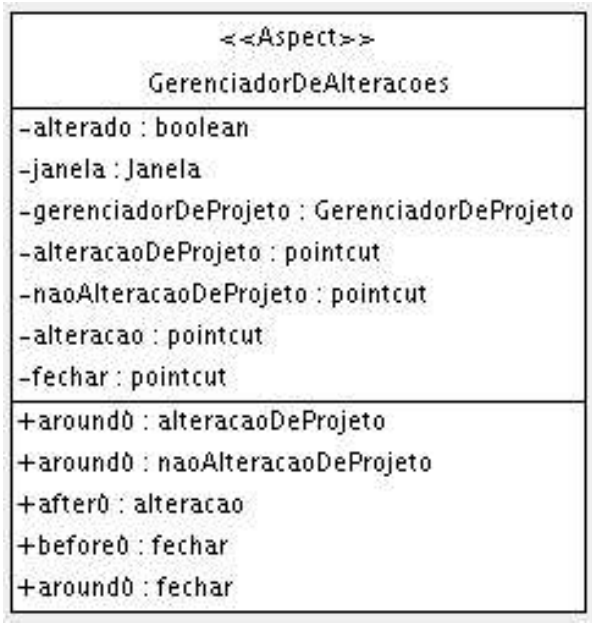
Conclusão

- AspectWerkz
  - Ferramenta mais “madura”.
  - Apresenta um conceito incompleto de Aspecto como unidade de encapsulamento.
  - HotDeploy “fraco”.

PROSE

- Apresenta Aspecto como unidade de encapsulamento completa.
- Está em um estágio inicial.
- Não provê suporte a Introductions.
- Maior legibilidade, porque não utiliza XML (nem programação declarativa), o que também diminui o seu poder.

A Importância do Conceito de Aspectos



Referências

- Aspect Oriented Software Development – www.aosd.net
- JBossAOP – www.jboss.org/developers/projects/jboss/aop
- AspectWerkz – aspectwerkz.codehaus.org
- PROSE – prose.ethz.ch