

Weaving technologies

Comparison of different weaving approaches

Bio

Professional

- Siemens CT SE2
 - Eclipse RCP
- IBM Rational Reasearch
 - Eclipse JDT Refactoring

Studies

- TUHH
 - General engineering science
- HAW Hamburg
 - Computer Engineering

Overview

- Used Terminology
- Offline weaving (static weaving)
- Load-time weaving
- Online weaving (dynamic weaving)

Terminology

- Advice
 - Code that will be inserted at a joinpoint
 - Often defined as „regular“ Java code
- Joinpoint
 - Point in control flow of a application
 - before/after/around method/field access/constructor
- Pointcut
 - Collection of joinpoints
 - Often matched via regular expressions

Terminology

- Aspect
 - Collection of advices that are assigned to joinpoints using pointcuts
- Weaving
 - Process of applying an Aspect to an existing application
 - Usually involves modification of bytecode

Offline weaving

Offline weaving

- Bytecode information
- Bytecode manipulation tools
- General weaving process

Bytecode information

Example Application:

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello Welt");  
    }  
}
```

Bytecode Information

```
public static main(String[]) : void
L0 (0)
  LINENUMBER 5 L0
  GETSTATIC System.out : PrintStream
  LDC "Hello Welt"
  INVOKEVIRTUAL PrintStream.println(String) : void
L1 (4)
  LINENUMBER 6 L1
  RETURN
L2 (6)
  LOCALVARIABLE args String[] L0 L2 0
  MAXSTACK = 2
  MAXLOCALS = 1
```

Java Virtual Machine

- Stack driven
 - Invocation
 - Pushes parameters
 - (Pops result)
 - Execution
 - Pops parameters
 - Pushs result
- Execution Modes
 - Interpreted
 - Compiled
 - Baseline compiled
 - Optimised compiled

Demo with bytecode outline

Bytecode Modification

Structures that need updates

- Constant pool
 - String references to classes
 - Literals
- References to Labels
- MaxStack
- MaxLocals
- Exceptions table
- Attributes

BCEL Example

- Insert „before“ advice
- Insert „after“ advice

```
public class TestClass {  
    public static void main(String[] args) {  
        if (args.length == 0)  
            return;  
        System.out.print("World");  
    }  
}
```

BCEL Example

```
JavaClass testClass =  
    Repository.lookupClass("TestClass");  
Method[] methods = testClass.getMethods();  
ClassGen gen = new ClassGen(testClass);  
InstructionFactory factory = new  
    InstructionFactory(gen);
```

- Loading Class into special representation
- Can not modify already loaded classes!

BCEL Code print

Calling getCode on Method object

Code(max_stack = 2 , max_locals = 1 , code_length = 15)

0: aload_0

1: array length

2: ifne #6

5: **return**

6: getstatic java.lang.System.out Ljava/io/PrintStream ; (22)

9: ldc "World" (24)

11: invokevirtual java.io.PrintStream.print(Ljava/lang/String;)V (30)

14: **return**

BCEL Insert before

```
MethodGen mg = new
    MethodGen(method, TestClass.getClassName(), gen
        .getConstantPool());
InstructionList original = mg.getInstructionList();
InvokeInstruction before =
    factory.createInvoke(Aspect.class.getName(),
        "before", Type.VOID, new Type[0],
        Constants.INVOKESTATIC);
original.insert(before);
mg.setMaxLocals();
mg.setMaxStack();
Method newMethod = mg.getMethod();
```

BCEL Code print

Calling getCode again

```
Code(max_stack = 2, max_locals = 1, code_length = 18)
0:  invokestatic  offline.general.bcel.Aspect.before ()V (39)
3:  aload_0
4:  arraylength
5:  ifne        #9
8:  return
9:  getstatic   java.lang.System.out Ljava/io/PrintStream; (22)
12: ldc        "World" (24)
14: invokevirtual
      java.io.PrintStream.print (Ljava/lang/String;)V (30)
17: return
```

BCEL Insert after

```
InstructionList original = mg.getInstructionList();
InstructionHandle[] handles =
    original.getInstructionHandles();
for (InstructionHandle handle : handles) {
    if (handle.getInstruction()
        .equals(InstructionConstants.RETURN)) {
        InvokeInstruction after = factory.createInvoke(
            Aspect.class.getName(), "after",
            Type.VOID,
            new Type[0], Constants.INVOKESTATIC);
        original.insert(handle, after);
    }
}
```

Load-time weaving

Load-Time weaving

- How it works
- Possibilities
- Restrictions

Load time weaving?!

- Bytecode manipulation same as offline
 - Often BCEL, ASM..
- Configuration often through XML

LTW in AspectJ

```
public class TestClass {
    public static void main(String[] args) throws
        Exception {
        ClassLoader parent =
            ClassLoader.getSystemClassLoader();
        ClassLoader cl = new
            WeavingURLClassLoader(parent);
        Class<?> loadClass =
            cl.loadClass("net.kbsvn.ltw.HelloClass");
        ISayHello hc=(ISayHello) loadClass.newInstance();
        hc.sayHello();
    }
}

public interface ISayHello {
    public void sayHello();
}
```

Possibilities for ClassLoader

- Redefine Systemclassloader through define
 - -Djava.system.class.loader
- Replace ClassLoader through hotswap
 - JVM needs to run in debug modus
- Replace ClassLoader in rt.jar
 - -Xbootclasspath

Restrictions and side-effects

- Only classes loaded through ClassLoader
 - Due to parent first paradigm to be woven code may not lie within ClassPath
- Classes already loaded can not be woven
- Only aspects that are loaded can be applied
 - No loading of aspects during runtime
- No classes from rt.jar
 - Loaded through bootstrap CL

Online Weaving

- Through Proxies
 - Spring
 - JBoss
- Through JVMDI&JVMTI
 - Prose
 - Spring (agent)
 - JBoss (agent)
- Through deep JVM support
 - Steamloom
 - JRocket?

Proxies

- In Java since 1.3
- During runtime created
- Use interfaces for method signatures
- Dispatch invocations to single method
 - `InvocationHandler#invoke()`

Proxies advantages

- Advice lookup during runtime
 - Very easy to make dynamic
- Supported in every JVM
- Support before, after, around with modification of result

Proxies disadvantage

- Classes are not the same
 - Problems with annotations
 - getClass() name differs
- No support for field access
- Small performance penalty
- No Call support

JVMDI&JVMTI

- Support through
 - Breakpoint (JVMDI)
 - General invocation notification (JVMTI)
 - Bytecode redefinition (JVMTI & JVMDI)
- Requires some support in JVM
 - Most JVMs support both
- Requires native Code
 - Portability issues
- Requires access to JVM commandline
 - Not always easy in J2EE
- Have access to all classes

JVMDI&JVMTI

- Lowers overall performance
 - JVMDI needs Debug mode of JVM
 - No penalty for client VM
 - Up to 20% penalty for server VM
 - Redefinition of Class break optimisation level
- Only partial support for Intertype definition
- Not more than 1 supported
 - In Java6 this restriction has been changed

Deep JVM support

- Steamloom
 - Works for jikes RVM on linux
 - Performance close to aspectJ static
 - Faster! for cflow
 - Faster Advice instance lookup
 - No double book-keeping
 - Required information already included in Class objects
- JRocket
 - Not much known
 - Subscriber kind of weaving

Ready to go, or questions?

Any questions, or ready for deep JVM support?

Jikes RVM

- Research Virtual Machine
 - Written in Java
 - Selfhosted
 - No interpreted mode
 - 2 JIT Compiler
 - Baseline JIT
 - Adaptive Optimisation System

Jikes RVM

- Method are replaced by lazy compilation stubs
- Lazy compilation stub triggers compilation of real bytecode
 - Done using Baseline compiler
 - Stupid but blazing fast
- Optimised Compiler performs basic optimisation
- Adaptive Compiler compiles profile based optimised code

Steamloom

- Replaces Compilers
 - Additional information about method
 - Inline places
 - BAT compatible Class structure
- Advice deployment
 - Weaves method bytecode using BAT
 - Flag method for recompilation
- Advice undeployment
 - Unweave method bytecode using BAT
 - Flag method for recompilation

Steamloom API

- Interfaced using regular Java

