Euxx

Thursday November 29, 2007

Merging class methods with ASM / Java : 22:13 EST

While ago I wrote paper for AOSD'08, which among other examples, showed how to merge bytecode from two classes.

The class to merge can be loaded into a ClassNode, and then the main class can be visited with the MergeAdapter that will also take ClassNode as a parameter:

```
public class MergeAdapter extends ClassAdapter {
  private ClassNode cn;
  private String cname;

  public MergeAdapter(ClassVisitor cv, ClassNode cn) {
    super(cv);
    this.cn = cn;
  }

  public void visit(int version, int access, String name,
      String signature, String superName, String[] interfaces) {
    super.visit(version, access, name, signature, superName, interfaces);
    this.cname = name;
  }

  public void visitEnd() {
    for (Iterator it = cn.fields.iterator(); it.hasNext();) {
      ((FieldNode) it.next()).accept(this);
    }
    for (Iterator it = cn.methods.iterator(); it.hasNext();) {
      MethodNode mn = (MethodNode) it.next();
      String[] exceptions = new String[mn.exceptions.size()];
      mn.exceptions.toArray(exceptions);
      MethodVisitor mv = cv.visitMethod(mn.access, mn.name,
          mn.desc, mn.signature, exceptions);
      mn.instructions.resetLabels();
      mn.accept(new RemappingMethodAdapter(mn.access,
          mn.desc, mv, new SimpleRemapper(cname, cn.name)));
    }
    super.visitEnd();
  }
}
```

The heavy duty work is done by the RemappingMethodAdapter, which is part of the ASM 3.1 API.

The above code is a very generic and sope of the AOSD'08 paper didn't allow me to cover all the corner cases, the real applications, such as dead sexy Terracotta or good old AspectWerkz have to deal with. For example, case when methods or constructors are present in both classes. Merging two methods is easy enough, but merging code from constructors gets really tricky.

The problem is that JVM doesn't allow to access class state before "super()" is called. So, when merging code from two constructors, call to "super()" need to be skipped. That call can appear really hairy in the bytecode, and generally isolation of that bytecode require data flow analysis. I won't bug you with all low level details, but the good news is that since ASM 2.1, there is AdviceAdapter, which can hide this complexity and allow to process bytecode in one pass.

Basically, when you need to do something after call to the "super()" or "on method enter" for regular methods, you can subclass AdviceAdapter adapter and overwrite onMethodEnter() method. Similarly, for doing something on "method exit", you can overwrite onMethodExit():

```
class FooAdapter extends AdviceAdapter {

  public FooAdapter(MethodVisitor mv, String owner,
      int access, String name, String desc, String callBack) {
    super( mv, access, name, desc);
  }
```

```
  protected void onMethodEnter() {
    ...
  }

  protected void onMethodExit(int opcode) {
    ...
  }
}
```

Now, for merging constructors the action is to skip all the code until right after super call. This can be also done with AdviceAdapter. Credit for this variant goes to Tim Eck (hey Tim, you really should start blogging!):

```
class SkippingAdapter extends AdviceAdapter {

  public SkippingAdapter(MethodNode mn, int access, String name, String desc) {
    super(mn, access, name, desc);
  }

  protected void onMethodEnter() {
    ((MethodNode) mv).instructions.clear();
  }
}
```

In the above example method bytecode will be loaded into MethodNode, so it clears method instructions captured before the "super()" call.

But what if you don't need method bytecode in memory? Well, AdviceAdapter can still help with that. Here is how:

```
class SkippingAdapter extends MethodAdapter {
  MethodAdapter nextAdapter;

  public SkippingAdapter(MethodVisitor mv, int access, String name, String desc)
    super(new AdviceAdapter(new EmptyVisitor(), access, name, desc) {
        protected void onMethodEnter() {
           SkippingAdapter.this.mv = nextAdapter;
        }
      });
    nextAdapter = mv;
  }
}
```

The idea is to redirect all events to EmptyVisitor and then flip the MethodVisitor right after "super()" is called.

Permalink : : Trackback 

COMMENTS:

POST A COMMENT:

Comments are closed for this entry.

Content and code from this blog by Eugene Kuleshov is licensed under a Creative Commons Attribution 3.0 License. You are free to built upon anything from this blog, but should attribute the original author (Eugene Kuleshov) and give a link to the corresponding blog entry.