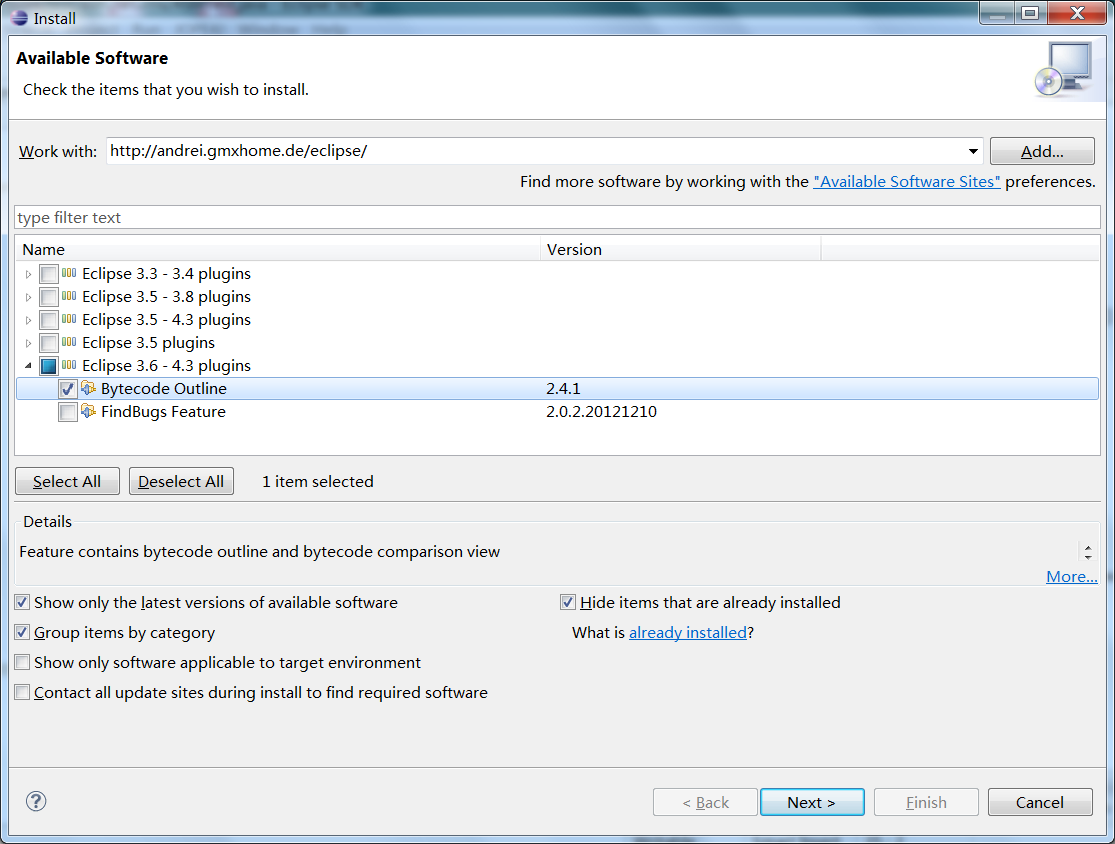
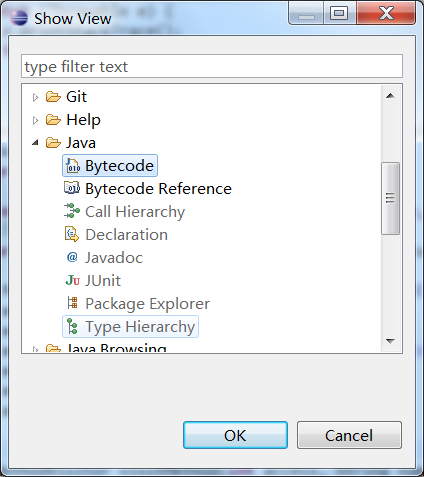
本文是《[深入字节码 -- 使用 ASM 实现 AOP](http://my.oschina.net/u/1166271/blog/162796)》的后续博文。在上一篇文章中介绍了如何使用 ASM 动态安插代码到类中，从而简单实现 Aop。文章得到了广大朋友好评，我也希望可以不负众望继续写出可以得到大家认可的更多相关文章。废话不多进入正题。

    古语有云“工欲善其事，必先利其器”。由于 JVM 对字节码十分敏感修改过程中稍微有一丝错误都会导致虚拟机错误，而想要排查错误却是一件比较困难的事情。再加上后面的博文将会很大程度上依赖 “ASM-Bytecode” 工具。因此我觉得有必要在深入制定字节码之前介绍一下如何使用 “ASM-Bytecode” 。

    首先安装Eclipse插件，插件的地址为：“http://andrei.gmxhome.de/eclipse/” 我的 Eclipse 版本为 3.7。  


安装完成之后重启 Eclipse ，打开菜单 Window -> Show View -> Other... 在分类中选择 Bytecode 视图  


    为了测试其功能随便创建一个工程并新建一个 HalloWord 程序，在 Eclipse 中打开 “HalloWord.java” 程序查看 Bytecode 视图，你会得到下面这样的代码。  
    （注意：由于Bytecode会自动感知 Eclipse 编辑器中光标位置从而确定生成的代码范围因此初学者建议将光标放到 “main” 方法中）

// access flags 0x9

public static main(String[]) : void

L0

LINENUMBER 22 L0

GETSTATIC System.out : PrintStream

LDC "Hallo Word"

INVOKEVIRTUAL PrintStream.println(String) : void

L1

LINENUMBER 23 L1

RETURN

L2

LOCALVARIABLE args String[] L0 L2 0

MAXSTACK = 2

MAXLOCALS = 1

    这是 ASM 为我们生成的 “main” 方法字节码指令。点击 “Bytecode”  视图右上角红色的 “ASM” 按钮。ASM便会为我们生成想要的 ASM 代码。

{

mv = cw.visitMethod(ACC\_PUBLIC + ACC\_STATIC, "main", "([Ljava/lang/String;)V", null, null);

mv.visitCode();

Label l0 = new Label();

mv.visitLabel(l0);

mv.visitLineNumber(22, l0);

mv.visitFieldInsn(GETSTATIC, "java/lang/System", "out", "Ljava/io/PrintStream;");

mv.visitLdcInsn("Hallo Word");

mv.visitMethodInsn(INVOKEVIRTUAL, "java/io/PrintStream", "println", "(Ljava/lang/String;)V");

Label l1 = new Label();

mv.visitLabel(l1);

mv.visitLineNumber(23, l1);

mv.visitInsn(RETURN);

Label l2 = new Label();

mv.visitLabel(l2);

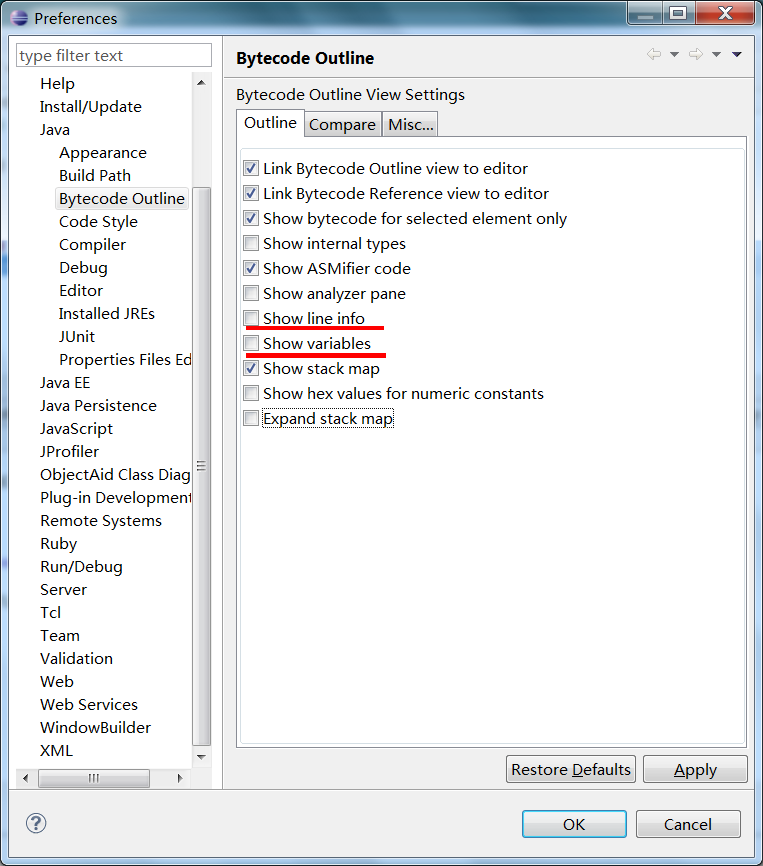
mv.visitLocalVariable("args", "[Ljava/lang/String;", null, l0, l2, 0);

mv.visitMaxs(2, 1);

mv.visitEnd();

}

    “Bytecode ” 程序总会为我们生成很多不必要的代码，为此打开 Window -> Preferences 找到 Bytecode Outline 选项关闭 “Show line info”、 “Show variables” 两个选项。这两个选项分别是用来生成行号代码和本地变量表代码。这样做可以大大减少所要分析的内容，不过即便如此 “Bytecode Outline” 的生成代码中依然保留了很多不必要的 “垃圾” 。 



    最后得到如下精简的 ASM 生成内容：4，5，9，10，12，13 行代码仍然是垃圾。很可惜 “ Bytecode Outline ” 只帮我们去掉了 “visitLineNumber” 这样的代码，其他两行并未给予处理。只要记得这个是目前是用来表示行号即可。

{

mv = cw.visitMethod(ACC\_PUBLIC + ACC\_STATIC, "main", "([Ljava/lang/String;)V", null, null);

mv.visitCode();

Label l0 = new Label();

mv.visitLabel(l0);

mv.visitFieldInsn(GETSTATIC, "java/lang/System", "out", "Ljava/io/PrintStream;");

mv.visitLdcInsn("Hallo Word");

mv.visitMethodInsn(INVOKEVIRTUAL, "java/io/PrintStream", "println", "(Ljava/lang/String;)V");

Label l1 = new Label();

mv.visitLabel(l1);

mv.visitInsn(RETURN);

Label l2 = new Label();

mv.visitLabel(l2);

mv.visitMaxs(2, 1);

mv.visitEnd();

}

    要注意的是虽然借助 “Bytecode Outline” 我们只需要提供一份代码模板即可生成各种 ASM 代码，但是，**切莫生成过于复杂的代码。**

    到这里使用 “Bytecode Outline” 生成 ASM 代码部分就介绍这么多，在下一篇文章中将重点介绍 ASM 核心接口方法。