Atitit java的动态编译 开发效率

面的demo, 先构建一个类的源文件, 再使用流写入到磁盘中, 接着调用编译api, 编译磁盘上的源文件, 最后使用反射调加载并调用编译后的字节码. 代码如下:

编译jsp调用

动态编译的两种做法:

通过Runtime调用javac,启动新的进程去操作(6.0之前,不是真正的动态编译)

Runtime run = Runtime.getRuntime();

Process process = run.exec("javac -cp d:/myjava/Helloworld.java")

通过JavaCompiler动态编译

​

通过JavaCompiler动态编译

JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();

int result = compiler.run(null, null, null, "f:/HelloWorld.java");

Parameters:

in "standard" input; use System.in if null

out "standard" output; use System.out if null

err "standard" error; use System.err if null

2.动态运行编译好的类

通过Runtime.getRuntime()运行启动新的进程运行

JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();

int result = compiler.run(null, null, null, "f:/HelloWorld.java");

System.out.println(result==0?"编译成功":"编译失败");

Runtime run = Runtime.getRuntime();

Process process = run.exec("java -cp f: HelloWorld");

BufferedReader w = new BufferedReader(new InputStreamReader(process.getInputStream()));

System.out.println(w.readLine());

通过反射运行编译好的类

MethodUtils.*invokeStaticMethod*(cls, "main", (Object)new String[]{});

m.invoke(null, (Object)new String[]{});//静态方法不用谢调用的对象

//加Object强制转换的原因

//由于可变参数是JDK5.0之后才有　m.invoke(null, new String[]{"23","34"});

//编译器会把它编译成m.invoke(null,"23","34");的格式,会发生参数不匹配的问题

//带数组的参数都这样做

package script;

import java.io.File;

import java.io.IOException;

import javax.tools.JavaCompiler;

import javax.tools.ToolProvider;

import org.apache.commons.io.FileUtils;

import org.apache.commons.lang3.reflect.MethodUtils;

public class JavaUti {

public static void main(String[] **args**) throws Exception {

String javafile = "C:\\Users\\jun\\eclipse-workspace\\deadlockchk\\src\\main\\java\\script\\Bean2.java";

String out\_classesPathDir = "c:\\outClasses";

FileUtils.*forceMkdir*(new File(out\_classesPathDir));

*invokeJava*(javafile, out\_classesPathDir);

}

private static void invokeJava(String **javafile**, String **out\_classesPathDir**) throws Exception {

JavaCompiler javac = ToolProvider.*getSystemJavaCompiler*();

File javaFile=new File(**javafile**);

//JavaCompiler最核心的方法是run, 通过这个方法编译java源文件, 前3个参数传null时,

//分别使用标准输入/输出/错误流来 处理输入和编译输出. 使用编译参数-d指定字节码输出目录.

int compileResult = javac.run(null, null, null, "-d", new File(**out\_classesPathDir**).getAbsolutePath(), javaFile.getAbsolutePath());

//run方法的返回值: 0-表示编译成功, 否则表示编译失败

if(compileResult != 0) {

System.***err***.println("编译失败!!");

return;

}

String className = "script.Bean2";

Class cls = Class.*forName*(className);

MethodUtils.*invokeStaticMethod*(cls, "main", (Object)new String[]{});

return;

}

}

// 当前编译器

JavaCompiler cmp = ToolProvider.getSystemJavaCompiler();

//Java 标准文件管理器

StandardJavaFileManager fm = cmp.getStandardFileManager(null,null,null);

//Java 文件对象

JavaFileObject jfo = new StringJavaObject(clsName,sourceStr);

// 编译参数，类似于javac <options> 中的options

List<String> optionsList = new ArrayList<String>();

// 编译文件的存放地方，注意：此处是为Eclipse 工具特设的

optionsList.addAll(Arrays.asList("-d","./bin"));

// 要编译的单元

List<JavaFileObject> jfos = Arrays.asList(jfo);

// 设置编译环境

Java Compiler.CompilationTask task = cmp.getTask(null, fm, null,optionsList,null,jfos);

// 编译成功

if(task.call()){

// 生成对象

Object obj = Class.forName(clsName).newInstance();

Class<? extends Object> cls = obj.getClass();

// 调用sayHello 方法

Method m = cls.getMethod(methodName, String.class);

String str = (String) m.invoke(obj, "Dynamic Compilation");

System.out.println(str);

}

Java 的动态编译对源提供了多个渠道。比如， 可以是字符串（ 例子中就是字符串），可以是文本文件，也可以是编译过的字节码文件（.class 文件），甚至可以是存放在数据库中的明文代码或是字节码。汇总成一句话， 只要是符合Java 规范的就都可以在运行期动态加载， 其实现方式就是实现JavaFileObject 接口， 重写getCharContent、openInputStream、openOutputStream，或者实现JDK 已经提供的两个SimpleJavaFileObject、ForwardingJavaFileObject，具体代码可以参考上个例子。