Javassist 简单应用小结

概述

Javassist 是一款字节码编辑工具，可以直接编辑和生成 Java 生成的字节码，以达到对.class 文件进行动态修改的效果。熟练使用这套工具，可以让 Java 编程更接近与动态语言编程。

下面一个方法的目的是获取一个类加载器（ClassLoader），以加载指定的.jar 或.class 文件，在之后的代码中会使用到。

[java] view plaincopyprint?

```
1.  private static ClassLoader getLocaleClassLoader() throws Exception {
2.      List<URL> classPathURLs = new ArrayList<>();
3.      // 加载.class 文件路径
4.      classPathURLs.add(classesPath.toURI().toURL());
5.
6.      // 获取所有的 jar 文件
7.      File[] jarFiles = libPath.listFiles(new FilenameFilter() {
8.          @Override
9.          public boolean accept(File dir, String name) {
10.             return name.endsWith(".jar");
11.         }
12.     });
13.     Assert.assertFalse(ObjectHelper.isArrayNullOrEmpty(jarFiles));
14.
15.     // 将 jar 文件路径写入集合
16.     for (File jarFile : jarFiles) {
17.         classPathURLs.add(jarFile.toURI().toURL());
18.     }
19.
20.     // 实例化类加载器
21.     return new URLClassLoader(classPathURLs.toArray(new URL[classPathURLs
    .size()]));
22. }
```

获取类型信息

[java] view plaincopyprint?

```
1.  @Test
2.  public void test() throws NotFoundException {
3.      // 获取默认类型池对象
4.      ClassPool classPool = ClassPool.getDefault();
5.
6.      // 获取指定的类型
7.      CtClass ctClass = classPool.get("java.lang.String");
8.
9.      System.out.println(ctClass.getName());  // 获取类名
10.     System.out.println("\tpackage " + ctClass.getPackageName());    // 获
    取包名
11.     System.out.print("\t" + Modifier.toString(ctClass.getModifiers()) + "
     class " + ctClass.getSimpleName());   // 获取限定符和简要类名
12.     System.out.print(" extends " + ctClass.getSuperclass().getName());  /
    / 获取超类
13.     // 获取接口
14.     if (ctClass.getInterfaces() != null) {
15.         System.out.print(" implements ");
16.         boolean first = true;
17.         for (CtClass c : ctClass.getInterfaces()) {
18.             if (first) {
19.                 first = false;
20.             } else {
21.                 System.out.print(", ");
22.             }
23.             System.out.print(c.getName());
24.         }
25.     }
26.     System.out.println();
27. }
```

修改类方法

[java] view plaincopyprint?

```
1.  @Test
2.  public void test() throws Exception {
3.      // 获取本地类加载器
4.      ClassLoader classLoader = getLocaleClassLoader();
5.      // 获取要修改的类
6.      Class<?> clazz = classLoader.loadClass("edu.alvin.reflect.TestLib");


7.
8.      // 实例化类型池对象
9.      ClassPool classPool = ClassPool.getDefault();
10.     // 设置类搜索路径
11.     classPool.appendClassPath(new ClassClassPath(clazz));
12.     // 从类型池中读取指定类型
13.     CtClass ctClass = classPool.get(clazz.getName());
14.
15.     // 获取 String 类型参数集合
16.     CtClass[] paramTypes = {classPool.get(String.class.getName())};
17.     // 获取指定方法名称
18.     CtMethod method = ctClass.getDeclaredMethod("show", paramTypes);
19.     // 赋值方法到新方法中
20.     CtMethod newMethod = CtNewMethod.copy(method, ctClass, null);
21.     // 修改源方法名称
22.     String oldName = method.getName() + "$Impl";
23.     method.setName(oldName);
24.
25.     // 修改原方法
26.     newMethod.setBody("{System.out.println(\"执行前
    \");" + oldName + "($$);System.out.println(\"执行后\");}");
27.     // 将新方法添加到类中
28.     ctClass.addMethod(newMethod);
29.
30.     // 加载重新编译的类
```

```
31.     clazz = ctClass.toClass();       // 注意，这一行会将类冻结，无法在对字节码
    进行编辑
32.     // 执行方法
33.     clazz.getMethod("show", String.class).invoke(clazz.newInstance(), "he
    llo");
34.     ctClass.defrost();  // 解冻一个类，对应 freeze 方法
35. }
```

动态创建类

[java] view plaincopyprint?

```
1.  @Test
2.  public void test() throws Exception {
3.      ClassPool classPool = ClassPool.getDefault();
4.
5.      // 创建一个类
6.      CtClass ctClass = classPool.makeClass("edu.alvin.reflect.DynamiClass"
    );
7.      // 为类型设置接口
8.      //ctClass.setInterfaces(new CtClass[] {classPool.get(Runnable.class.g
    etName())});
9.
10.     // 为类型设置字段
11.     CtField field = new CtField(classPool.get(String.class.getName()), "v
    alue", ctClass);
12.     field.setModifiers(Modifier.PRIVATE);
13.     // 添加 getter 和 setter 方法
14.     ctClass.addMethod(CtNewMethod.setter("setValue", field));
15.     ctClass.addMethod(CtNewMethod.getter("getValue", field));
16.     ctClass.addField(field);
17.
18.     // 为类设置构造器
19.     // 无参构造器
20.     CtConstructor constructor = new CtConstructor(null, ctClass);
```

```
21.     constructor.setModifiers(Modifier.PUBLIC);
22.     constructor.setBody("{}");
23.     ctClass.addConstructor(constructor);
24.     // 参数构造器
25.     constructor = new CtConstructor(new CtClass[] {classPool.get(String.c
    lass.getName())}, ctClass);
26.     constructor.setModifiers(Modifier.PUBLIC);
27.     constructor.setBody("{this.value=$1;}");
28.     ctClass.addConstructor(constructor);
29.
30.     // 为类设置方法
31.     CtMethod method = new CtMethod(CtClass.voidType, "run", null, ctClass
    );
32.     method.setModifiers(Modifier.PUBLIC);
33.     method.setBody("{System.out.println(\"执行结果\" + this.value);}");
34.     ctClass.addMethod(method);
35.
36.     // 加载和执行生成的类
37.     Class<?> clazz = ctClass.toClass();
38.     Object obj = clazz.newInstance();
39.     clazz.getMethod("setValue", String.class).invoke(obj, "hello");
40.     clazz.getMethod("run").invoke(obj);
41.
42.     obj = clazz.getConstructor(String.class).newInstance("OK");
43.     clazz.getMethod("run").invoke(obj);
44. }
```

创建代理类

[java] view plaincopyprint?

```
1.  @Test
2.  public void test() throws Exception {
3.      // 实例化代理类工厂
4.      ProxyFactory factory = new ProxyFactory();
```

```
5.
6.      //设置父类，ProxyFactory 将会动态生成一个类，继承该父类
7.      factory.setSuperclass(TestProxy.class);
8.
9.      //设置过滤器，判断哪些方法调用需要被拦截
10.     factory.setFilter(new MethodFilter() {
11.         @Override
12.         public boolean isHandled(Method m) {
13.             return m.getName().startsWith("get");
14.         }
15.     });
16.
17.     Class<?> clazz = factory.createClass();
18.     TestProxy proxy = (TestProxy) clazz.newInstance();
19.     ((ProxyObject)proxy).setHandler(new MethodHandler() {
20.         @Override
21.         public Object invoke(Object self, Method thisMethod, Method proce
    ed, Object[] args) throws Throwable {
22.             //拦截后前置处理，改写 name 属性的内容
23.             //实际情况可根据需求修改
24.             System.out.println(thisMethod.getName() + "被调用");
25.             try {
26.                 Object ret = proceed.invoke(self, args);
27.                 System.out.println("返回值: " + ret);
28.                 return ret;
29.             } finally {
30.                 System.out.println(thisMethod.getName() + "调用完毕");
31.             }
32.         }
33.     });
34.
35.     proxy.setName("Alvin");
36.     proxy.setValue("1000");
37.     proxy.getName();
38.     proxy.getValue();
```

```
39. }
```

其中，TestProxy 类内容如下：

[java] view plaincopyprint?

```
1.  public class TestProxy {
2.      private String name;
3.      private String value;
4.
5.      public String getName() {
6.          return name;
7.      }
8.      public void setName(String name) {
9.          this.name = name;
10.     }
11.     public String getValue() {
12.         return value;
13.     }
14.     public void setValue(String value) {
15.         this.value = value;
16.     }
17. }
```

获取方法名称

[java] view plaincopyprint?

```
1.  @Test
2.  public void test() throws Exception {
3.      // 获取本地类加载器
4.      ClassLoader classLoader = getLocaleClassLoader();
5.      // 获取要修改的类
6.      Class<?> clazz = classLoader.loadClass("edu.alvin.reflect.TestLib");


7.
8.      // 实例化类型池
```

```
9.      ClassPool classPool = ClassPool.getDefault();
10.     classPool.appendClassPath(new ClassClassPath(clazz));
11.     CtClass ctClass = classPool.get(clazz.getName());
12.
13.     // 获取方法
14.     CtMethod method = ctClass.getDeclaredMethod("show", ObjectHelper.argu
    mentsToArray(CtClass.class, classPool.get("java.lang.String")));
15.     // 判断是否为静态方法
16.     int staticIndex = Modifier.isStatic(method.getModifiers()) ? 0 : 1;


17.
18.     // 获取方法的参数
19.     MethodInfo methodInfo = method.getMethodInfo();
20.     CodeAttribute codeAttribute = methodInfo.getCodeAttribute();
21.     LocalVariableAttribute localVariableAttribute = (LocalVariableAttribu
    te)codeAttribute.getAttribute(LocalVariableAttribute.tag);
22.
23.     for (int i = 0; i < method.getParameterTypes().length; i++) {
24.         System.out.println("第" + (i + 1) + "个参数名称
    为: " + localVariableAttribute.variableName(staticIndex + i));
25.     }
26. }
```

关于“获取方法名称”，其主要作用是：当 Java 虚拟机加载.class 文件后，会将类方法“去名称化”，即丢弃掉方法形参的参数名，而是用形参的序列号来传递参数。如果要通过 Java 反射获取参数的参数名，则必须在编辑是指定“保留参数名称”。Javassist 则不存在这个问题，对于任意方法，都能正确的获取其参数的参数名。

Spring MVC 就是通过方法参数将请求参数进行注入的，这一点比 struts2 MVC 要方便很多，Spring 也是借助了 Javassist 来实现这一点的。