javassist使用全解析

Java 字节码以二进制的形式存储在 .class 文件中，每一个 .class 文件包含一个 Java 类或接口。Javaassist 就是一个用来 处理 Java 字节码的类库。它可以在一个已经编译好的类中添加新的方法，或者是修改已有的方法，并且不需要对字节码方面有深入的了解。同时也可以去生成一个新的类对象，通过完全手动的方式。

1. 使用 Javassist 创建一个 class 文件#

首先需要引入jar包：

```
Copy<dependency>
  <groupId>org.javassist</groupId>
  <artifactId>javassist</artifactId>
  <version>3.25.0-GA</version>
</dependency>
```

编写创建对象的类：

```
Copypackage com.rickiyang.learn.javassist;

import javassist.*;

/**
 * @author rickiyang
 * @date 2019-08-06
 * @Desc
 */
public class CreatePerson {

    /**
     * 创建一个Person 对象
     *
     * @throws Exception
     */
    public static void createPseson() throws Exception {
        ClassPool pool = ClassPool.getDefault();

        // 1. 创建一个空类
        CtClass cc = pool.makeClass("com.rickiyang.learn.javassist.Person");

        // 2. 新增一个字段 private String name;
        // 字段名为name
        CtField param = new CtField(pool.get("java.lang.String"), "name", cc);
        // 访问级别是 private
        param.setModifiers(Modifier.PRIVATE);
        // 初始值是 "xiaoming"
        cc.addField(param, CtField.Initializer.constant("xiaoming"));

        // 3. 生成 getter、setter 方法
        cc.addMethod(CtNewMethod.setter("setName", param));
        cc.addMethod(CtNewMethod.getter("getName", param));
```

```
        // 4. 添加无参的构造函数
        CtConstructor cons = new CtConstructor(new CtClass[]{}, cc);
        cons.setBody("{name = \"xiaohong\";}");
        cc.addConstructor(cons);

        // 5. 添加有参的构造函数
        cons = new CtConstructor(new CtClass[]{pool.get("java.lang.String")},
cc);
        // $0=this / $1,$2,$3... 代表方法参数
        cons.setBody("{$0.name = $1;}");
        cc.addConstructor(cons);

        // 6. 创建一个名为printName方法，无参数，无返回值，输出name值
        CtMethod ctMethod = new CtMethod(CtClass.voidType, "printName", new
CtClass[]{}, cc);
        ctMethod.setModifiers(Modifier.PUBLIC);
        ctMethod.setBody("{System.out.println(name);}");
        cc.addMethod(ctMethod);

        //这里会将这个创建的类对象编译为.class文件
        cc.writeFile("/Users/yangyue/workspace/springboot-learn/java-
agent/src/main/java/");
    }

    public static void main(String[] args) {
        try {
            createPseson();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

执行上面的 main 函数之后，会在指定的目录内生成 Person.class 文件：

```
Copy//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by Fernflower decompiler)
//

package com.rickiyang.learn.javassist;

public class Person {
    private String name = "xiaoming";

    public void setName(String var1) {
        this.name = var1;
    }

    public String getName() {
        return this.name;
    }

    public Person() {
        this.name = "xiaohong";
    }
```

```
    public Person(String var1) {
        this.name = var1;
    }

    public void printName() {
        System.out.println(this.name);
    }
}
```

跟咱们预想的一样。

在 Javassist 中，类 `Javaassit.CtClass` 表示 class 文件。一个 GtClass (编译时类）对象可以处理一个 class 文件，`ClassPool` 是 `CtClass` 对象的容器。它按需读取类文件来构造 `CtClass` 对象，并且保存 `CtClass` 对象以便以后使用。

需要注意的是 ClassPool 会在内存中维护所有被它创建过的 CtClass，当 CtClass 数量过多时，会占用大量的内存，API中给出的解决方案是 **有意识的调用 `CtClass` 的 `detach()` 方法以释放内存。**

`ClassPool` 需要关注的方法：

1. getDefault : 返回默认的 `ClassPool` 是单例模式的，一般通过该方法创建我们的ClassPool；
2. appendClassPath, insertClassPath : 将一个 `ClassPath` 加到类搜索路径的末尾位置 或 插入到起始位置。通常通过该方法写入额外的类搜索路径，以解决多个类加载器环境中找不到类的尴尬；
3. toClass : 将修改后的CtClass加载至当前线程的上下文类加载器中，CtClass的 `toClass` 方法是通过调用本方法实现。**需要注意的是一旦调用该方法，则无法继续修改已经被加载的class**；
4. get , getCtClass : 根据类路径名获取该类的CtClass对象，用于后续的编辑。

`CtClass` 需要关注的方法：

1. freeze : 冻结一个类，使其不可修改；
2. isFrozen : 判断一个类是否已被冻结；
3. prune : 删除类不必要的属性，以减少内存占用。调用该方法后，许多方法无法将无法正常使用，慎用；
4. defrost : 解冻一个类，使其可以被修改。如果事先知道一个类会被defrost， 则禁止调用 prune 方法；
5. detach : 将该class从ClassPool中删除；
6. writeFile : 根据CtClass生成 `.class` 文件；
7. toClass : 通过类加载器加载该CtClass。

上面我们创建一个新的方法使用了 `CtMethod` 类。CtMthod代表类中的某个方法，可以通过CtClass提供的API获取或者CtNewMethod新建，通过CtMethod对象可以实现对方法的修改。

`CtMethod` 中的一些重要方法：

1. insertBefore : 在方法的起始位置插入代码；
2. insterAfter : 在方法的所有 return 语句前插入代码以确保语句能够被执行，除非遇到exception；
3. insertAt : 在指定的位置插入代码；
4. setBody : 将方法的内容设置为要写入的代码，当方法被 abstract修饰时，该修饰符被移除；
5. make : 创建一个新的方法。

注意到在上面代码中的：setBody()的时候我们使用了一些符号：

```
Copy// $0=this / $1,$2,$3... 代表方法参数
cons.setBody("{$0.name = $1;}");
```

具体还有很多的符号可以使用，但是不同符号在不同的场景下会有不同的含义，所以在这里就不在赘述，可以看javassist 的说明文档。http://www.javassist.org/tutorial/tutorial2.html

2. 调用生成的类对象#

1. 通过反射的方式调用

上面的案例是创建一个类对象然后输出该对象编译完之后的 .class 文件。那如果我们想调用生成的类对象中的属性或者方法应该怎么去做呢？javassist也提供了相应的api，生成类对象的代码还是和第一段一样，将最后写入文件的代码替换为如下：

```
Copy// 这里不写入文件，直接实例化
Object person = cc.toClass().newInstance();
// 设置值
Method setName = person.getClass().getMethod("setName", String.class);
setName.invoke(person, "cunhua");
// 输出值
Method execute = person.getClass().getMethod("printName");
execute.invoke(person);
```

然后执行main方法就可以看到调用了 `printName` 方法。

2. 通过读取 .class 文件的方式调用

```
CopyClassPool pool = ClassPool.getDefault();
// 设置类路径
pool.appendClassPath("/Users/yangyue/workspace/springboot-learn/java-agent/src/main/java/");
CtClass ctClass = pool.get("com.rickiyang.learn.javassist.Person");
Object person = ctClass.toClass().newInstance();
//  ...... 下面和通过反射的方式一样去使用
```

3. 通过接口的方式

上面两种其实都是通过反射的方式去调用，问题在于我们的工程中其实并没有这个类对象，所以反射的方式比较麻烦，并且开销也很大。那么如果你的类对象可以抽象为一些方法得合集，就可以考虑为该类生成一个接口类。这样在 `newInstance()` 的时候我们就可以强转为接口，可以将反射的那一套省略掉了。

还拿上面的 `Person` 类来说，新建一个 `PersonI` 接口类：

```
Copypackage com.rickiyang.learn.javassist;

/**
 * @author rickiyang
 * @date 2019-08-07
 * @Desc
 */
public interface PersonI {

    void setName(String name);

    String getName();

    void printName();

}
```

实现部分的代码如下：

```
CopyClassPool pool = ClassPool.getDefault();
pool.appendClassPath("/Users/yangyue/workspace/springboot-learn/java-
agent/src/main/java/");

// 获取接口
CtClass codeClassI = pool.get("com.rickiyang.learn.javassist.PersonI");
// 获取上面生成的类
CtClass ctClass = pool.get("com.rickiyang.learn.javassist.Person");
// 使代码生成的类，实现 PersonI 接口
ctClass.setInterfaces(new CtClass[]{codeClassI});

// 以下通过接口直接调用 强转
PersonI person = (PersonI)ctClass.toClass().newInstance();
System.out.println(person.getName());
person.setName("xiaolv");
person.printName();
```

使用起来很轻松。

2. 修改现有的类对象#

前面说到新增一个类对象。这个使用场景目前还没有遇到过，一般会遇到的使用场景应该是修改已有的类。比如常见的日志切面，权限切面。我们利用javassist来实现这个功能。

有如下类对象：

```
Copypackage com.rickiyang.learn.javassist;

/**
 * @author rickiyang
 * @date 2019-08-07
 * @Desc
 */
public class PersonService {

    public void getPerson(){
        System.out.println("get Person");
    }

    public void personFly(){
        System.out.println("oh my god,I can fly");
    }
}
```

然后对他进行修改：

```
Copypackage com.rickiyang.learn.javassist;

import javassist.ClassPool;
import javassist.CtClass;
import javassist.CtMethod;
import javassist.Modifier;

import java.lang.reflect.Method;

/**
```

```
 * @author rickiyang
 * @date 2019-08-07
 * @Desc
 */
public class UpdatePerson {

    public static void update() throws Exception {
        ClassPool pool = ClassPool.getDefault();
        CtClass cc = pool.get("com.rickiyang.learn.javassist.PersonService");

        CtMethod personFly = cc.getDeclaredMethod("personFly");
        personFly.insertBefore("System.out.println(\"起飞之前准备降落伞\");");
        personFly.insertAfter("System.out.println(\"成功落地。。。。\");");


        //新增一个方法
        CtMethod ctMethod = new CtMethod(CtClass.voidType, "joinFriend", new
CtClass[]{}, cc);
        ctMethod.setModifiers(Modifier.PUBLIC);
        ctMethod.setBody("{System.out.println(\"i want to be your friend\");}");
        cc.addMethod(ctMethod);

        Object person = cc.toClass().newInstance();
        // 调用 personFly 方法
        Method personFlyMethod = person.getClass().getMethod("personFly");
        personFlyMethod.invoke(person);
        //调用 joinFriend 方法
        Method execute = person.getClass().getMethod("joinFriend");
        execute.invoke(person);
    }

    public static void main(String[] args) {
        try {
            update();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

在`personFly`方法前后加上了打印日志。然后新增了一个方法`joinFriend`。执行main函数可以发现已经添加上了。

另外需要注意的是：上面的`insertBefore()`和`setBody()`中的语句，如果你是单行语句可以直接用双引号，但是有多行语句的情况下，你需要将多行语句用`{}`括起来。javassist只接受单个语句或用大括号括起来的语句块。