ASMSupport使用手册

*——java字节码操作类库*

版本：0.4

作者:温斯群

第1章 关于ASMSupport

1.1 简介

ASMSupport是一个java的字节码操作类库，底层采用的是ASM。那么相比于直接是用ASM，ASMSupport的优势在于它让开发者不需要了解底层指令的情况下能够快速的开发出动态生成字节码的业务。通过实践归结出以下几个asmsupport的优势：

1. 屏蔽了JVM底层指令。
2. 自动堆栈管理，无需像使用asm一般操作堆栈。
3. 采用匿名类的方式模拟java代码的各程序块，使得变量操作更安全。
4. 使用简单，上手快，可以快速的创建class

1.2 Maven·

<dependency>

<groupId>cn.wensiqun</groupId>

<artifactId>asmsupport</artifactId>

<version>0.4</version>

</dependency>

第2章 从Hello World开始

2.1 第一个ASMSupport程序

在介绍ASMSupport的一大篇长篇大论之前，先喝口咖啡，通过一个Hello World实例来感受下ASMSupport是如何创建一个类的。虽然你现在还不明白是如何使用asmsupport，但是Hello World嘛，你懂的。

先看下我们想要生成的代码具体是什么样的：

**public** **class** HelloWorldExample

{

**public** **static** **void** main(String[] args)

{

System.*out*.println("Hello World");

}

}

代码2-1

很简单的一段代码. 首先分析下操作流程，和我们常规编写代码一样，首先我需创建一个类，那么我们就需要一个类创建器，在ASMSupport中他是cn.wensiqun.asmsupport.creator.ClassCreator，我们通过new操作获取一个类创建器；随之而来的就是创建一个main方法，由于main方法是一个static类型的，那么我们将会调用他的createStaticMethod方法来帮助我们创建main方法；光有方法名没有方法的实际内容可不行，如果没有方法体，那么就是创建一个接口来，而这里我们还希望能够打印出Hello World呢，所以在调用createStaticMethod的时候我们会传递一个方法体对象cn.wensiqun.asmsupport.block.method.common.StaticMethodBody，这个对象才是核心，我们需要通过匿名类的方式来实现具体的方法体中的内容。那么如何使用ASMSupport去实现呢，代码送上：

ClassCreator creator = **new** ClassCreator(

Opcodes.*V1\_5*, Opcodes.*ACC\_PUBLIC* ,

"generated.helloworld.HelloWorldExample", **null**, **null**);

creator.createStaticMethod("main",

**new** AClass[]{AClassFactory.*getProductClass*(String[].**class**)},

**new** String[]{"args"}, **null**, **null**,

Opcodes.*ACC\_PUBLIC* + Opcodes.*ACC\_STATIC*,

**new** StaticMethodBody(){

@Override

**public** **void** generateBody(LocalVariable... argus) {

invoke(*systemOut*, "println", Value.*value*("Hello World"));

//don't forget return.

runReturn();

}

});

代码2-2

2.2 原理分析

上面我们分析过ASMSupport的操作流程，接下来我们从字节码层面上分析下ASMSupport的实现原理。首先我们先来看看上面main方法中的内容在字节码层面上是如何表示的：

GETSTATIC System.out

LDC "HelloWorld"

INVOKEIRTUAL PrintStream.println(String): void

RETURN

代码2-3

首先是获取一个System的一个静态成员变量out.然后再将”Hello World”压入操作栈，再调用out的println方法，并且将栈顶的”Hello World”传入println方法，最后再调用一个RETURN操作跳出。整个过程很简单，而前面三条指令在我们代码中表现的行为其实就是一句话System.out.println("Hello World!"),而后一条指令RETURN实际上就是main方法中没有显示的写明的return操作。所以在ASMSupport中为了更符合代码编写的习惯，将前三条指令封装成一个类cn.wensiqun.asmsupport.operators.method.StaticMethodInvo-ker。RETURN指令也封装成一个类cn.wensiqun.asmsupport.operators.Return.然后将这两个类放到当前方法的一个执行队列中。然后按照顺序执行队列中的各个对象，进而调用底层的ASM原始字节码操作实现方法中字节码的动态编写。下图能够简单描述这一过程。

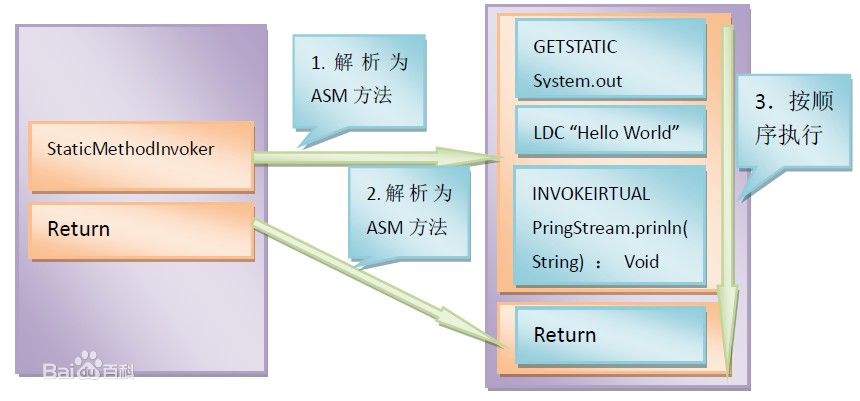


图2-1

第3章 概念篇

这一章主要介绍ASMSupport中的常用类和包结构。是一个比较枯燥的章节，可以将这一章当作是一个工具篇，在实际开发过程中遇到不明白的类能在此章找到相对应的解决方法。如果是初学者也可以在粗略了解本章中介绍的AClass,Value, ClassCreator, ProgramBlock之后直接进入第三章实战篇，在实战篇遇到不明白的类之后进入本章节再细节上了解该类。

3.1 cn.wensiqun.asmsupport包

asmsupport所有的类都在这个包或者其子包下。那么下面我们看看这个包下的结构：

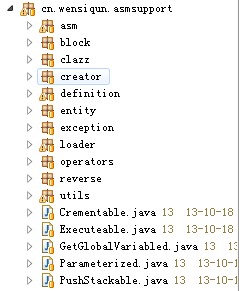


图3-1

在这里我们主要简单的介绍主要的几个包的定义。

* clazz包：asmsupport是模拟我们平时编写java的方式来实现的，所以在asmsupport中最基本的一个内容就是对class的描述。而这个包下就是存放着asmsupport对class的描述的类，包括了AClass,ArrayClass等等，将在后面详细介绍。
* definition包：这个包下我们定义了使用asmsupport的一些基本组件，包括方法，变量以及常量值等等。
* block包：此包下是我们对程序块的抽象，比如方法块，静态方法块，try...catch程序块，条件控制程序块等等。
* creator包：这个包我们定义了类创建器，类修改器，枚举类型创建器，方法创建器等等，这是我们使用asmsupport的入口。
* operators包：这个包下我们将所有的java操作都进行了抽象，比如算数运算，条件运算，方法调用，赋值等等。
* asm包：这个包下主要是存放一些asm底层操作的辅助类以及asm 3.3.1版本转向4.2版本的适配器，作为asmsupport的使用者可以不用了解这个包
* utils包：这里主要就是asmsupport的一些工具类。

3.2 clazz包

在我们平时编写java的时候我们有使用各种类，对于一般所使用的class我们将其抽象成cn.wensiqun.asmsupport.clazz.AClass。当然AClass仅仅是一个通用的类，在asmsupport中我们将其细分了如下一下类，如下图：

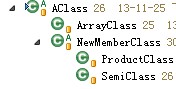


图3-2

我们可以看到AClass只是作为一个父类，其子类下还有ArrayClass，这是对数组类的抽象。NewMemberClass表示这个类是可以添加新的成员变量的，正常情况下，一些已存在的类是能够被修改的。比如我们需要动态修改某个类的时候，而这些已经存在的类我们称之为ProductClass，叫做成品类。被classloader加载过的类都是ProductClass类，与之对应的就是SemiClass，称之为半成品类，这就是我们使用asmsupport创建出来的类，当这个类被创建之后再次使用的时候就变成ProductClass类了。下面我们将介绍每个类。

3.2.1 AClass

1. **public** **abstract** **boolean** isArray() ：判断当前类是否是一个数组，这是一个抽象方法。返回类型是boolean类型，true表示是一个数组类。
2. **public** **abstract** **int** getDimension() ：如果当前类是个数组则返回这个数组类的维度，如果不是一个数组类则返回的是-1.
3. **public** String getPackage() ： 获取当前类的包名，返回的是个string类型。
4. **public** String getName() ：获取当前类的全名，包括了包名。
5. **public** **int** getModifiers() ：获取当前类的修饰符的值。可参考org.objectweb.asm.Opcodes类。
6. **public** **int** getVersion() ：获取当前被创建的类所使用的asm的版本。这个方法只适用于SemiClass类中。
7. **public** Class<?> getSuperClass() ： 获取当前类的父类。
8. **public** Class<?>[] getInterfaces() ：获取当前类的接口。
9. **public** **abstract** String getDescription() ： 获取当前类的在字节码层面上的描述，这是个抽象类，在其子类中实现。
10. **public** **boolean** isChildOrEqual(AClass cls) ： 判断当前类是否与传入的类相等或者是其子类.
11. **public** Type getType() ： 获取当前类的Type，Type是ASM的特有类。
12. **public** **abstract** GlobalVariableEntity getGlobalVariableEntity(-String name) : 获取当前类中的全局静态变量的Entity。
13. **public** **final** MethodEntity availableConstructor(AClass aclass, AClass[] parameterTypes) : 获取当前类中可调用的构造方法。由于访问控制修饰符，在不同的类中可能无法调用当前类中某个构造方法。这个方法就是可访问的构造方法，第一个参数表示在哪个类中调用构造方法，第二个参数表示构造方法的参数类型列表。
14. **public** **final** MethodEntity availableMethod(AClass where, String name, AClass[] parameterTypes) ： 获取可访问的方法。效果和之availableConstructor是一样的，第一个参数表示在哪个类中调用构造方法，第二个参数表示方法名，第二个参数表示构造方法的参数类型列表。
15. **public** MethodEntity getSuperMethod(String methodName, AClass[] parameterTypes) ： 获取可访问的父类中的方法，第一个参数表示方法名，第二个参数表示调用方法的参数类型列表。
16. **public** MethodEntity getSuperConstructor(AClass[] paraTypes) ： 获取可访问的父类中的构造方法。参数表示调用方法的参数类型列表。
17. **public** **abstract** **boolean** isPrimitive() : 判断当前类是否是基本类型。
18. **public** **final** **boolean** isInterface() ： 判断当前类是否是接口类型。
19. **public** **final** **boolean** isAbstract() ： 判断当前类是否是抽象类。
20. **public** **abstract** **int** getCastOrder() : 获取当前类的强制转换顺序。
21. **public** **final** Value getDefaultValue() ： 获取当前类的默认值，比如Object则返回的null，int类型则返回0. 但是其返回类型是Value。这是asmsupport对常量的特有封装，具体Value将在以后介绍。
22. **public** **abstract** **boolean** existStaticInitBlock() ： 获取当前类是否已经有了静态程序块。