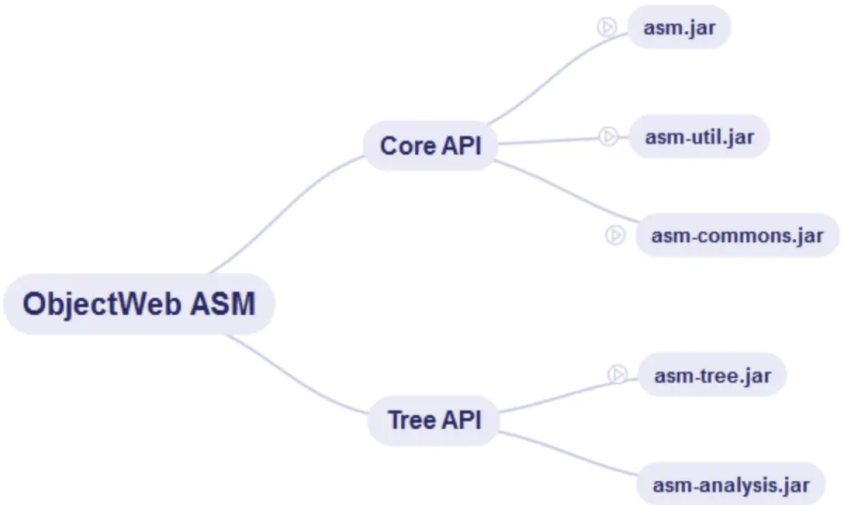
02 - ASM组成部分

舍是境界 关注
0.278 2022.01.08 07:39:54 字数 747 阅读 16

ASM的两个组成部分

从组成结构上来说，ASM分成两部分，一部分为Core API，另一部分为Tree API。

- 其中，Core API包括 `asm.jar` 、 `asm-util.jar` 和 `asm-commons.jar` ；
- 其中，Tree API包括 `asm-tree.jar` 和 `asm-analysis.jar` 。



ASM组成部分

从两者的关系来说，Core API是基础，而Tree API是在Core API的这个基础上构建起来的

Core API概览

这里是对 `asm.jar` 、 `asm-util.jar` 和 `asm-commons.jar` 文件里包含的主要类成员进行介绍。

asm.jar

在asm.jar文件中，一共包含了30多个类，会重点讲解10个核心类，其他的20多个主要起到“辅助”的作用，它们更多的倾向于是“幕后工作者”；

主要涉及到： `ClassVisitor` 、 `ClassWriter` 、 `FieldVisitor` 、 `FieldWriter` 、 `MethodVisitor` 、 `MethodWriter` 、 `Label` 、 `Opcodes` 、 `ClassReader` 和 `Type` 类。

其中最重要的三个类是： `ClassVisitor` 、 `ClassWriter` 、 `ClassReader` .

类关系图

这三个类的作用，可以简单理解成这样：

- `ClassReader` 类，负责读取.class文件里的内容，然后拆分成各个不同的部分。
- `ClassVisitor` 类，负责对.class文件中某一部分里的信息进行修改。
- `ClassWriter` 类，负责将各个不同的部分重新组合成一个完整的.class文件。

asm-util.jar

asm-util.jar主要包含的是一些工具类。这些类主要分成两种类型：Check开头和Trace开头。

- 以Check开头的类，主要负责检查（Check）生成的.class文件内容是否正确。
- 以Trace开头的类，主要负责将.class文件的内容打印成文字输出。根据输出的文字信息，可以探索或追踪（Trace）.class文件的内部信息。

asm-util.jar类示意图

asm-commons.jar

asm-commons.jar主要包含的是一些常用功能类。如下图所示，可以看到asm-commons.jar里面包含的具体类文件。

asm-commons.jar核心类

一个非常容易混淆的问题就是， `asm-util.jar` 与 `asm-commons.jar` 有什么区别呢？在asm-util.jar里，它提供的是通用性的功能，没有特别明确的应用场景；而在asm-commons.jar里，它提供

的功能，都是为解决某一种特定场景中出现的问题而提出的解决思路。

搭建ASM开发环境

- JDK版本：1.8.0_261
- Maven版本：3.8.1
- IDEA：2021.1.2 （Community Edition）

pom.xml

```
<properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>1.8</java.version>
        <maven.compiler.source>${java.version}</maven.compiler.source>
        <maven.compiler.target>${java.version}</maven.compiler.target>
        <asm.version>9.0</asm.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.ow2.asm</groupId>
            <artifactId>asm</artifactId>
            <version>${asm.version}</version>
        </dependency>
        <dependency>
            <groupId>org.ow2.asm</groupId>
            <artifactId>asm-commons</artifactId>
            <version>${asm.version}</version>
        </dependency>
        <dependency>
            <groupId>org.ow2.asm</groupId>
            <artifactId>asm-util</artifactId>
            <version>${asm.version}</version>
        </dependency>
        <dependency>
            <groupId>org.ow2.asm</groupId>
            <artifactId>asm-tree</artifactId>
            <version>${asm.version}</version>
        </dependency>
        <dependency>
            <groupId>org.ow2.asm</groupId>
            <artifactId>asm-analysis</artifactId>
            <version>${asm.version}</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Java Compiler -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>${java.version}</source>
                    <target>${java.version}</target>
                    <fork>true</fork>
                    <compilerArgs>
                        <arg>-g</arg>
                        <arg>-parameters</arg>
                    </compilerArgs>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

demo

```
package com.example;

import org.objectweb.asm.*;

public class HelloWorldDump implements Opcodes {

```

```
    public static byte[] dump() {
        ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_FRAMES);

        cw.visit(V1_8, ACC_PUBLIC | ACC_SUPER, "sample/HelloWorld", null, "java/lang/O

        {
            MethodVisitor mv1 = cw.visitMethod(ACC_PUBLIC, "<init>", "()V", null, null
            mv1.visitCode();
            mv1.visitVarInsn(ALOAD, 0);
            mv1.visitMethodInsn(INVOKESPECIAL, "java/lang/Object", "<init>", "()V", fa
            mv1.visitInsn(RETURN);
            mv1.visitMaxs(1, 1);
            mv1.visitEnd();
        }
        {
            MethodVisitor mv2 = cw.visitMethod(ACC_PUBLIC, "toString", "()Ljava/lang/S
            mv2.visitCode();
            mv2.visitLdcInsn("This is HelloWorld object.");
            mv2.visitInsn(ARETURN);
            mv2.visitMaxs(1, 1);
            mv2.visitEnd();
        }
        cw.visitEnd();

        return cw.toByteArray();
    }
}
```

```
package com.example;

public class MyClassLoader extends ClassLoader {
    @Override
    protected Class<?> findClass(String name) throws ClassNotFoundException {
        if ("sample.HelloWorld".equals(name)) {
            byte[] bytes = HelloWorldDump.dump();
            Class<?> clazz = defineClass(name, bytes, 0, bytes.length);
            return clazz;
        }

        throw new ClassNotFoundException("Class Not Found: " + name);
    }
}
```

```
package com.example;

public class HelloWorldRun {
    public static void main(String[] args) throws Exception {
        MyClassLoader classLoader = new MyClassLoader();
        Class<?> clazz = classLoader.loadClass("sample.HelloWorld");
        Object instance = clazz.newInstance();
        System.out.println(instance);
    }
}
```

输出结果

```
This is HelloWorld object.
```

小结

本文主要是对ASM的组成部分进行了介绍，内容总结如下：

- ASM由Core API和Tree API两个部分组成。
- Core API概览，就是对asm.jar、asm-commons.jar和asm-util.jar文件里包含的主要类成员进行介绍。
- 通过一个简单的示例，能够快速搭建起ASM的开发环境。

希望对你能有所帮助