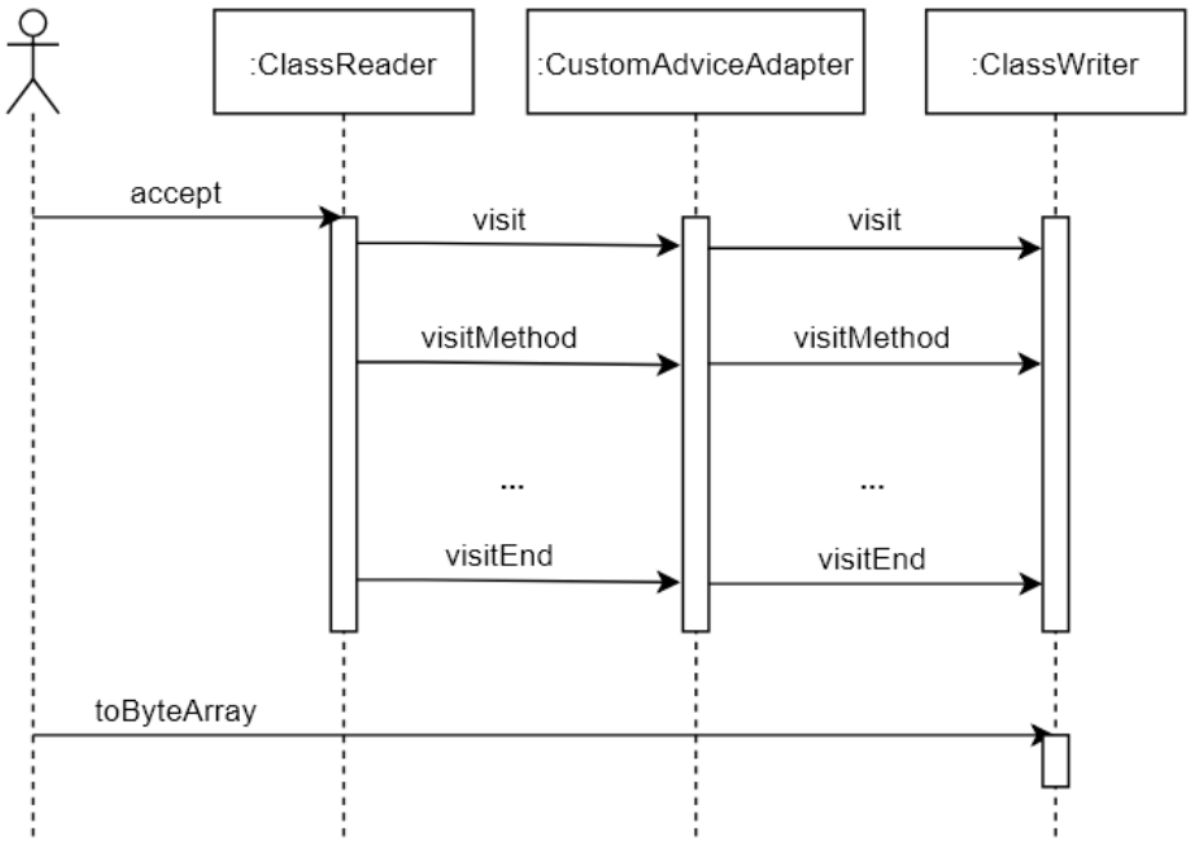
- asm作为一个字节码操作库，可以直接修改已经存在的class文件或者生成class文件
- 参考：https://blog.csdn.net/u013144863/article/details/107600341
- **上面这个链接最后举的例子就是咱们git上的demo，所以这个文章参考价值很高（个人认为）**
- jvm指令集整理：https://www.onesrc.cn/p/jvm-instruction-set-collation.html
- 核心API
  - ClassReader：对具体的 class 文件进行读取与解析
  - ClassVisitor、AdviceAdapter：可以访问class文件的各个部分，比如方法、变量、注解等，用于修改 class 文件。
  - ClassWriter：将修改后的class文件通过文件流的方式覆盖掉原来的 class 文件，从而实现 class 修改；
  - 

  - 整体流程：ASM 提供了一个类 `ClassReader` 可以方便地让我们对 `class` 文件进行读取与解析；ASM在 `ClassReader` 解析 `class` 文件过程中，解析到某一个结构就会通知到ClassVisitor 的相应方法（eg：解析到类方法时，就会回调`ClassVisitor.visitMethod` 方法）；

    可以通过更改 `ClassVisitor` 中相应结构方法返回值，实现对类的代码切入（eg：更改 ClassVisitor.visitMethod() 方法的默认返回值 MethodVisitor 实例，通过操作该自定义 MethodVisitor 从而实现对原方法的改写）；其它的结构遍历也如同 ClassVisitor；通过 ClassWriter 的 toByteArray() 方法，得到 class 文件的字节码内容，最后通过文件流写入方式覆盖掉原先的内容，实现 class`文件的改写。

- 字节码（深入理解书里面也有）：是Java虚拟机执行的一种指令格式。通俗来讲字节码就是经过 **javac** 命令编译之后生成的 class 文件。 class文件包含了 Java 虚拟机指令集和符号表以及若干其他的辅助信息。**可以通过 javap –c xxx.class终端命令来查看对应的字节码**
    - 字节码描述符：描述符的作用是描述字段的数据类型、方法的参数列表(包括数量、类型以及顺序)和返回值。对于基本数据类型(byte char double float int long short boolean)以及代表无返回值的void类型都用一个大写字符来表示，对象类型则用字符“L”加对象的全限定名来表示（即把包名所有“.”换成了“/” ），一般对象类型末尾都会加一个“;”来表示全限定名的结束。
    - Java 虚拟机栈描述的是 Java 方法执行的内存模型：每个方法在执行同时会创建一个 **栈帧** 用于存局部变量表、操作数栈、动态链接、方法返回地址等信息。每个方法从调用到执行完毕的过程，就对应着一个栈帧在虚拟机中从入栈到出栈的过程。每一个栈帧都包含了上述信息。一个线程中的方法调用链可能会很长，即会有很多栈帧。对于一个当前活动的线程中，只有位于线程栈顶的栈帧才是有效的，成为 **前栈帧（current stack Frame）** ，这个栈帧所关联的方法成为 **当前方法（current method）** 。
    - **局部变量表**：局部变量表是一组变量存储空间，用于存储方法参数（入参）和方法内部定义的局部变量。它的容量以容量槽为最小单位（slot）。虚拟机通过索引的定位方式使用这个局部变量表，从0开始，在非 static 方法中，**0 代表的是“this”，其余参数从1开始分配。**
    - **操作数栈**：它是一个后入先出的栈结构。当一个方法刚开始执行时，操作数栈是空的，执行过程中，会有各种字节码执行向操作数中写入和提取内容，也就是出栈和入栈的过程。
- # ASM Bytecode Outline 插件

```
public class test1 {

    private int a = 1;
    public int  add() {
        int b = 2;
        int c = a + b;

        return c;
    }

    public static void main(String[] args) {

    }

}
```

```
28  public add()I
29   L0
30    LINENUMBER 8 L0
31    ICONST_2
32    ISTORE 1
33   L1
34    LINENUMBER 9 L1
35    ALOAD 0
36    GETFIELD test1/test1.a : I
37    ILOAD 1
38    IADD
39    ISTORE 2
40   L2
41    LINENUMBER 11 L2
42    ILOAD 2
43    IRETURN
44   L3
45    LOCALVARIABLE this Ltest1/test1; L0 L3 0
46    LOCALVARIABLE b I L1 L3 1
47    LOCALVARIABLE c I L2 L3 2
48    MAXSTACK = 2
49    MAXLOCALS = 3
```

    - 看到当前类的asm字节码
- Q：我记得顾世辰同学提过一个问题，就是当我们使用asm插桩的时候会不会改变当前类的状态，然后我现在的理解是，尽管会，但是我们可以通过查看字节码来区分插桩前后的区别，从而直接把这些新插入的代码剔除，当然这只是我的一个猜想，逻辑上行得通，但是怎样简便实现我还在探索