目录

一、相关说明

1.1、jvm 及字节码

1.1.1、JVM 是 Java Virtual Machine（Java 虚拟机）的缩写。Java 语言使用 Java 虚拟机屏蔽了与具体平台相关的信息，使得 Java 语言编译程序只需生成在 Java 虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改地运行。Java 虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。这就是 Java 的能够“一次编译，到处运行”的原因。

(本段摘录自百度百科 https://baike.baidu.com/item/JVM/2902369?fr=aladdin)

1.1.2、JVM 及 java 字节码的相关技术参考了以下文档：

A、《Java 虚拟机：JVM 高级特性与最佳实践.周志明》，部分截图摘录自该文档。

B、The JavaTM Virtual Machine Specification
https://docs.oracle.com/javase/specs/jvms/se6/html/VMSpecTOC.doc.html

C、The class File Format
https://docs.oracle.com/javase/specs/jvms/se6/html/ClassFile.doc.html#74353

D、Opcode Mnemonics by Opcode
https://docs.oracle.com/javase/specs/jvms/se6/html/Mnemonics.doc.html

E、Java 虚拟机字节码指令
http://blog.csdn.net/wangxf_8341/article/details/50402525

1.2、相关工具

1.2.1、UltraEdit 查看、修改二进制文件。
1.2.2、jclasslib、反编译.class 文件，二进制信息将以 java 字节码指令助记符显示。
1.2.3、7zip、修改 jar 包中文件。

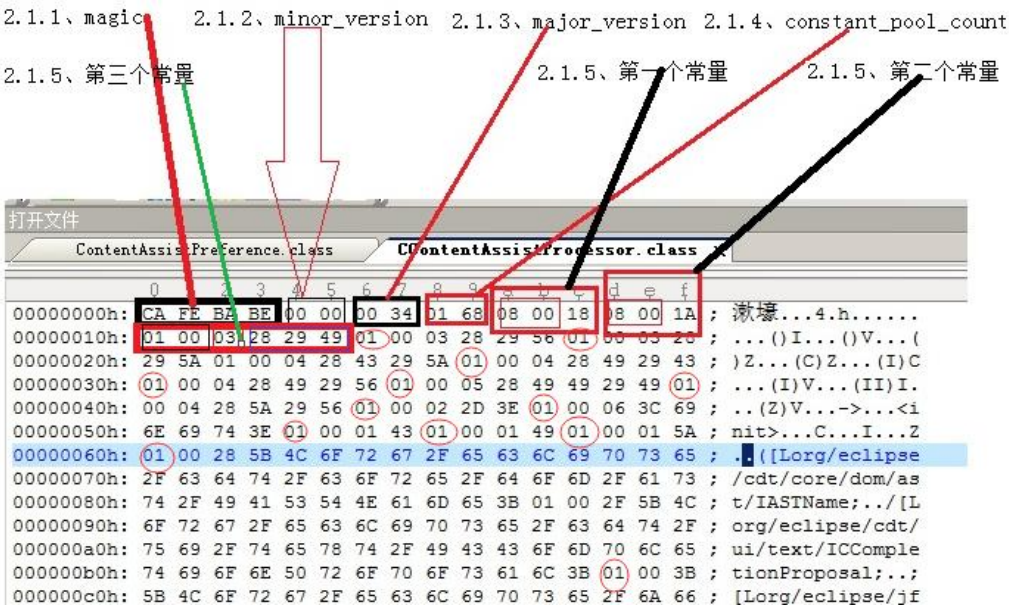
二、CContentAssistProcessor.class 文件代码分析

2.1、Class 字节码文件格式

Class 文件格式

类 型	名 称	数 量
u4	magic	1
u2	minor_version	1
u2	major_version	1
u2	constant_pool_count	1
cp_info	constant_pool	constant_pool_count-1
u2	access_flags	1
u2	this_class	1
u2	super_class	1
u2	interfaces_count	1
u2	interfaces	interfaces_count
u2	fields_count	1
field_info	fields	fields_count
u2	methods_count	1
method_info	methods	methods_count
u2	attributes_count	1
attribute_info	attributes	attributes_count

2.1.1、最开始的 4 个字节是 Class 文件的魔数(magic)标识 0XCAFEBABE。
2.1.2、第二个信息是第 5~8 字节，存储着 Class 文件的版本号。第 5~6 个字节是次版本号;第 7~8 个字节是主版本。(下图的主版本号为 0x34=52，次版本号为 0 表示是 jdk1.8.0)
2.1.3、紧接着 2 字节 constant_pool_count 是常量池数目。常量池数目标识着接下来的常量池有多少个常量。(数目为 0x0168=360，常量池索引计数从 1 开始，索引值为 1~359 个数量)



CContentAssistProcessor.class 字节码开始到部分常量池的字节

Class 文件版本号

Class 文件版本号

编译器版本	-target 参数	十六进制版本号	十进制版本号
JDK 1.1.8	不能带 target 参数	00 03 00 2D	45.3
JDK 1.2.2	不带（默认为 -target 1.1）	00 03 00 2D	45.3
JDK 1.2.2	-target 1.2	00 00 00 2E	46.0
JDK 1.3.1_19	不带（默认为 -target 1.1）	00 03 00 2D	45.3
JDK 1.3.1_19	-target 1.3	00 00 00 2F	47.0
JDK 1.4.2_10	不带（默认为 -target 1.2）	00 00 00 2E	46.0
JDK 1.4.2_10	-target 1.4	00 00 00 30	48.0
JDK 1.5.0_11	不带（默认为 -target 1.5）	00 00 00 31	49.0
JDK 1.5.0_11	-target 1.4 -source 1.4	00 00 00 30	48.0
JDK 1.6.0_01	不带（默认为 -target 1.6）	00 00 00 32	50.0
JDK 1.6.0_01	-target 1.5	00 00 00 31	49.0
JDK 1.6.0_01	-target 1.4 -source 1.4	00 00 00 30	48.0
JDK 1.7.0	不带（默认为 -target 1.7）	00 00 00 33	51.0
JDK 1.7.0	-target 1.6	00 00 00 32	50.0
JDK 1.7.0	-target 1.4 -source 1.4	00 00 00 30	48.0

2.2、常量池分析

2.2.1、常量池的项目类型

常量池有 11 种常量类型，所有类型的常量第一个信息，都是 1 字节的标志(tag，取值为 1~12)，标识着当前常量为哪种常量类型。如下图所示：

常量池的项目类型

类　型	标志	描　述
CONSTANT_Utf8_info	1	UTF-8 编码的字符串
CONSTANT_Integer_info	3	整型字面量
CONSTANT_Float_info	4	浮点型字面量
CONSTANT_Long_info	5	长整型字面量
CONSTANT_Double_info	6	双精度浮点型字面量
CONSTANT_Class_info	7	类或接口的符号引用
CONSTANT_String_info	8	字符串类型字面量
CONSTANT_Fieldref_info	9	字段的符号引用
CONSTANT_Methodref_info	10	类中方法的符号引用
CONSTANT_InterfaceMethodref_info	11	接口中方法的符号引用
CONSTANT_NameAndType_info	12	字段或方法的部分符号引用

CONSTANT_Utf8_info 型常量的结构

类型	名称	数量
u1	tag	1
u2	length	1
u1	bytes	length

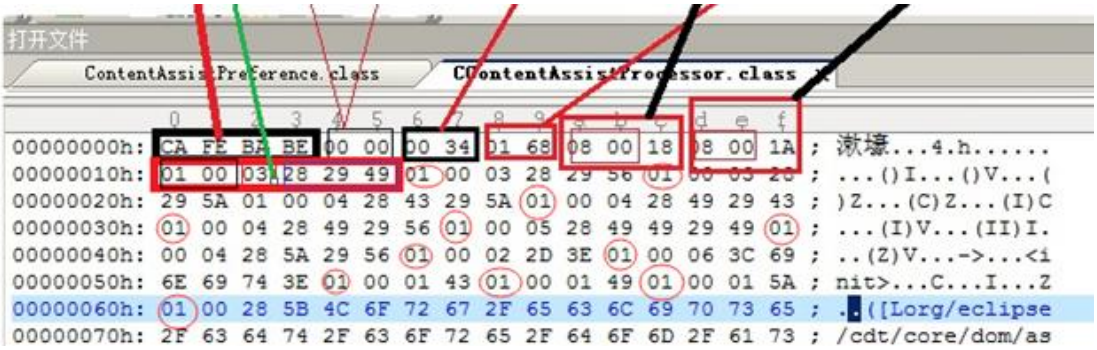
CONSTANT_Class_info 型常量的结构

类型	名称	数量
u1	tag	1
u2	name_index	1

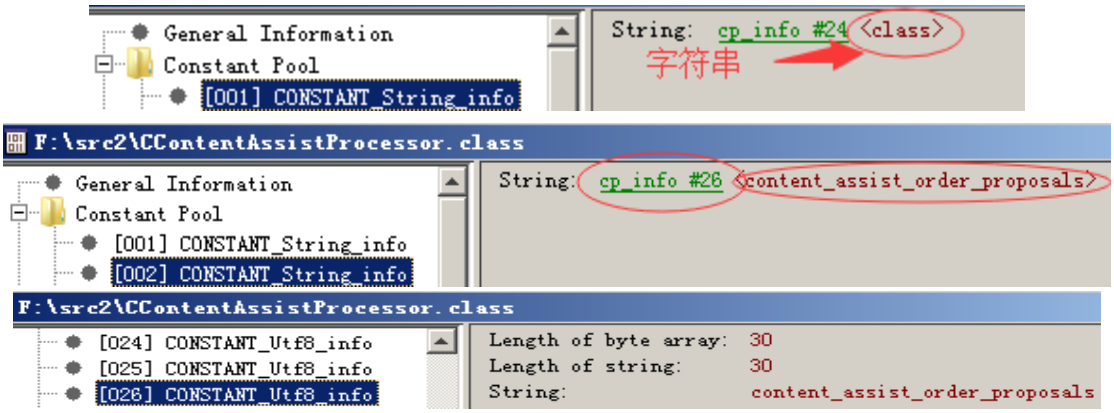
CONSTANT_String_info 型常量的结构

CONSTANT_String_info	tag	u1	值为 8
	index	u2	指向字符串字面量的索引

2.2.2、CContentAssistProcessor.class 的前 3 个常量分析



⑴、第 1(0x080018)与第 2(0x08001A)个常量都以 0x08 开头， tag=0x08 是 CONSTANT_String_info 型常量，0x18=24 指向索引号为 24 的 CONSTANT_Utf8_info 字符串字面量的索引；而 0x1A=26 指向索引号为 26 的 CONSTANT_Utf8_info 字符串字面量的索引。



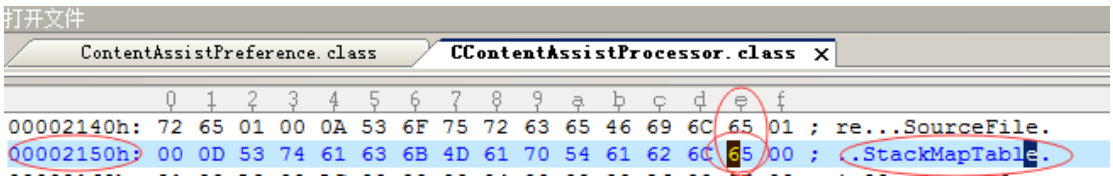
⑵、第 3 个常量的字节为 0x010003282949，以 0x01 开头，tag=0x01 是 CONSTANT_Utf8_info 类型常量，数目为 0x03，字符串为 0x282949 = "()I"，是一个返回值为 int 的方法描述符。

⑶、接下来的第 4~10 个常量都为 CONSTANT_Utf8_info 类型常量，都是某个方法的描述符。

2.2.3、CContentAssistProcessor.class 最后一个常量的定位。

⑴、通过 jclasslib 软件，我们可以很容易的知道，第 359 个常量也为 CONSTANT_Utf8_info 类型常量，其字符串为“StackMapTable”。

⑵、在 UltraEdit 打开的 CContentAssistProcessor.class 文档中，切换为字符显示，查找“StackMapTable”，在整个文件中只有一个匹配项目。把光标定位到最后的"e"字符，然后切换为十六进制显示，得到其所在的字节为 0x0215e，如下图所示：



常量池分析到此为此，其他常量的分析类似。

2.2.4、常量池中的 11 种数据类型结构总表

常量	项目	类型	描述
CONSTANT_Utf8_info	tag	u1	值为 1
	length	u2	UTF-8 编码的字符串占用了字节数
	bytes	u1	长度为 length 的 UTF-8 编码的字符串
CONSTANT_Integer_info	tag	u1	值为 3
	bytes	u4	按照高位在前存储的 int 值
CONSTANT_Float_info	tag	u1	值为 4
	bytes	u4	按照高位在前存储的 float 值
CONSTANT_Long_info	tag	u1	值为 5
	bytes	u8	按照高位在前存储的 long 值
CONSTANT_Double_info	tag	u1	值为 6
	bytes	u8	按照高位在前存储的 double 值
CONSTANT_Class_info	tag	u1	值为 7
	index	u2	指向全限定名常量项的索引
CONSTANT_String_info	tag	u1	值为 8
	index	u2	指向字符串字面量的索引
CONSTANT_Fieldref_info	tag	u1	值为 9
	index	u2	指向声明字段的类或接口描述符 CONSTANT_Class_info 的索引项
	index	u2	指向字段描述符 CONSTANT_NameAndType 的索引项
CONSTANT_Methodref_info	tag	u1	值为 10
	index	u2	指向声明方法的类描述符 CONSTANT_Class_info 的索引项
	index	u2	指向名称及类型描述符 CONSTANT_NameAndType 的索引项
CONSTANT_InterfaceMethodref_info	tag	u1	值为 11
	index	u2	指向声明方法的接口描述符 CONSTANT_Class_info 的索引项
	index	u2	指向名称及类型描述符 CONSTANT_NameAndType 的索引项
CONSTANT_NameAndType_info	tag	u1	值为 12
	index	u2	指向该字段或方法名称常量项的索引
	index	u2	指向该字段或方法描述符常量项的索引

2.3、访问标志

紧跟常量池后面的 2 个字节为访问标志(access_flags)，其值为 0x0021，表示 public 类型的类。

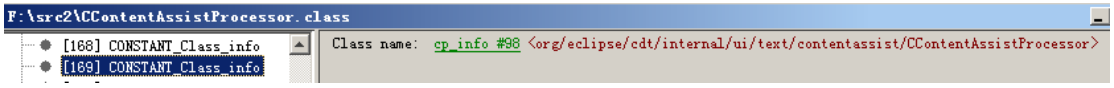
访问标志

标志名称	标志值	含　义
ACC_PUBLIC	0x0001	是否为 public 类型
ACC_FINAL	0x0010	是否被声明为 final，只有类可设置
ACC_SUPER	0x0020	是否允许使用 invokespecial 字节码指令，JDK 1.2 之后编译出来的类的这个标志为真
ACC_INTERFACE	0x0200	标识这是一个接口
ACC_ABSTRACT	0x0400	是否为 abstract 类型，对于接口或抽象类来说，此标志值为真，其他类值为假
ACC_SYNTHETIC	0x1000	标识这个类并非由用户代码产生的
ACC_ANNOTATION	0x2000	标识这是一个注解
ACC_ENUM	0x4000	标识这是一个枚举

2.4、类/父类索引与接口索引集合

接着的 3 个或以上的 u2 类型共 6 个或以上的字节(从索引为 0x2161 字节开始)的信息为类索引、父类索引，及接口索引集合。

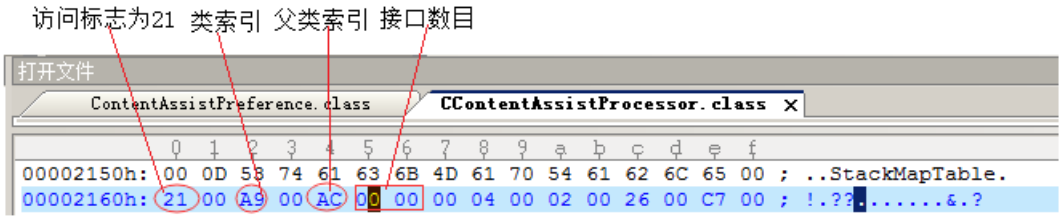
2.4.1、类索引 this_class，其值为 0x00A9 = 169，对应着常量池索引为 169，类型为 CONSTANT_Class_info 的常量，指向路径为“org/eclipse/cdt/internal/ui/text/contentassist/CContentAssistProcessor”的类。



2.4.2、父类索引 super_class 其值为 0x00AC = 172，对应着常量池索引为 172，类型为 CONSTANT_Class_info 的常量，指向路径为“org/eclipse/cdt/internal/ui/text/contentassist/ContentAssistProcessor”的类。

2.4.3、接口索引数目与接口索引集合

指向 interfaces_count 是索引为 0x2165 开始的 2 个字节，其值为 0，表示该类没有实现任何接口。故后面的索引表容量为 0，即 interfaces 不再占用字节。



2.5、字段表集合

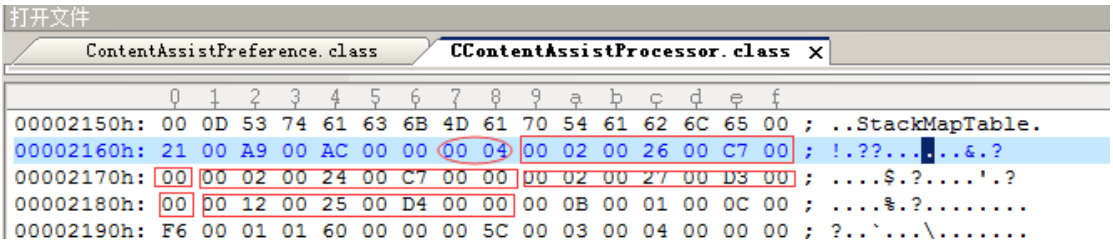
接下来的 2 字节 0x0004 表示 fields_count 数目为 4。字段表包含了类变量(即全局静态变量)或实例变量(即全局非静态变量)。字段表结构与访问标志(access_flags)如下：

字段表结构

类 型	名 称	数 量
u2	access_flags	1
u2	name_index	1
u2	descriptor_index	1
u2	attributes_count	1
attribute_info	attributes	attributes_count

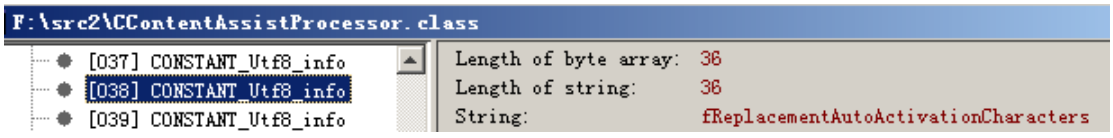
字段访问标志

标志名称	标志值	含 义
ACC_PUBLIC	0x0001	字段是否 public
ACC_PRIVATE	0x0002	字段是否 pravate
ACC_PROTECTED	0x0004	字段是否 protected
ACC_STATIC	0x0008	字段是否 static
ACC_FINAL	0x0010	字段是否 final
ACC_VOLATILE	0x0040	字段是否 volatile
ACC_TRANSIENT	0x0080	字段是否 transient
ACC_SYNTHETIC	0x1000	字段是否由编译器自动产生的
ACC_ENUM	0x4000	字段是否 enum



2.5.1、分析第 1 个字段

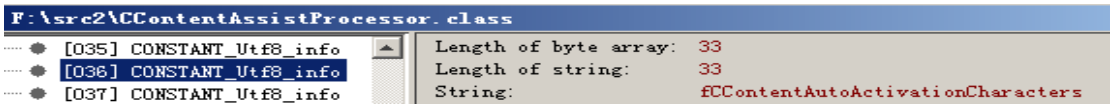
access_flag=2,private; name_index=0x26=38; descriptor_index=0xC7=199; attributes_count=0
第 38 个与 199 个常量分别如下图所示



即该成员变量为：private ActivationSet fReplacementAutoActivationCharacters;

2.5.2、分析第 2 个字段

access_flag=2,private; name_index=0x24=36; descriptor_index=0xC7=199; attributes_count=0
第 36 个常量如下图所示



即该成员变量为：private ActivationSet fCContentAutoActivationCharacters;

2.5.3、分析第 3 个字段

access_flag=2,private;　name_index=0x27=39["fValidator"];
descriptor_index=0xD3=211["IContextInformationValidator "];　attributes_count=0
即该成员变量为：private　IContextInformationValidator　fValidator;
2.5.4、分析第 4 个字段
access_flag=0x12,private final;　name_index=0x25=37["fEditor"];
descriptor_index=0xD4=212["IEditorPart"];　attributes_count=0
即该成员变量为：private　final　IEditorPart　fEditor;

2.6、方法表相关介绍

方法表的内容与字段表类似，表结构的描述几乎是完全一致的。索引为 0x2189 开始的 2 个字节，其值为 0x000B=11，表示该类有 11 个方法。方法表结构与方法访问标志如下：

方法表结构

类　型	名　称	数　量
u2	access_flags	1
u2	name_index	1
u2	descriptor_index	1
u2	attributes_count	1
attribute_info	attributes	attributes_count

方法访问标志

标志名称	标志值	含　义
ACC_PUBLIC	0x0001	方法是否为 public
ACC_PRIVATE	0x0002	方法是否为 private
ACC_PROTECTED	0x0004	方法是否为 protected
ACC_STATIC	0x0008	方法是否为 static
ACC_FINAL	0x0010	方法是否为 final
ACC_ SYNCHRONIZED	0x0020	方法是否为 synchronized
ACC_BRIDGE	0x0040	方法是否是由编译器产生的桥接方法
ACC_VARARGS	0x0080	方法是否接受不定参数
ACC_ NATIVE	0x0100	方法是否为 native
ACC_ABSTRACT	0x0400	方法是否为 abstract
ACC_STRICT	0x0800	方法是否为 strictfp
ACC_SYNTHETIC	0x1000	方法是否是由编译器自动产生的

2.7、属性表相关介绍

方法表中将会出现“Code”等属性，其中“Code”属性用于存在字节码指令，其结构如下：

虚拟机规范预定义的属性

属性名称	使用位置	含　义
Code	方法表	Java 代码编译成的字节码指令
ConstantValue	字段表	final 关键字定义的常量值
Deprecated	类、方法表、字段表	被声明为 deprecated 的方法和字段
Exceptions	方法表	方法抛出的异常
InnerClasses	类文件	内部类列表
LineNumberTable	Code 属性	Java 源码的行号与字节码指令的对应关系
LocalVariableTable	Code 属性	方法的局部变量描述
SourceFile	类文件	源文件名称
Synthetic	类、方法表、字段表	标识方法或字段为编译器自动生成的

属性表结构

类　型	名　称	数　量
u2	attribute_name_index	1
u4	attribute_lenght	1
u1	info	attribute_lenght

Code 属性表的结构

类　型	名　称	数　量
u2	attribute_name_index	1
u4	attribute_length	1
u2	max_stack	1
u2	max_locals	1
u4	code_length	1
u1	code	code_length
u2	exception_table_length	1
exception_info	exception_table	exception_table_length
u2	attributes_count	1
attribute_info	attributes	attributes_count

2.8、方法与字段的描述符

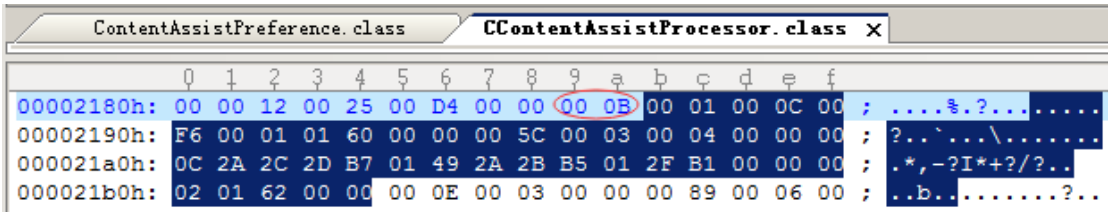
描述符是用来描述字段的数据类型，方法的参数列表(数量、类型及顺序)与返回值。基本数据类型用 1 个大写字母表示。对象类型用‘L’加对象对应的类的全限定名来表示。而数组类型，每一维度，用一个前置的“[”表示，如 int[][]将表示为[[I。

描述符标识字符含义

标识字符	含　义
B	基本类型 byte
C	基本类型 char
D	基本类型 double
F	基本类型 float
I	基本类型 int
J	基本类型 long
S	基本类型 short
Z	基本类型 boolean
V[①]	特殊类型 void
L	对象类型，如 Ljava/lang/Object;

2.9、分析第 1 个方法

上面提到，该类一共有 11 个方法，第一个方法从索引为 0x218b 字节开始。



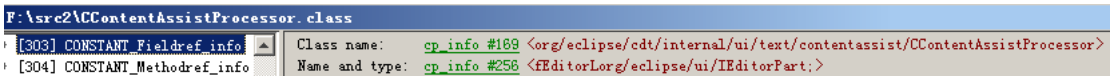
2.9.1、方法表属性

access_flag=1,public;　name_index=0x0c=12["<init>"]，这是实例构造器方法;
descriptor_index=0xF6=246
["(Lorg/eclipse/ui/ IEditorPart;　Lorg/eclipse/jface/text/contentassist/ ContentAssistant;Ljava/lang/String;)V"];
即方法头应为：
public CContentAssistProcessor(IEditorPart editor, ContentAssistant assistant, String part);
attributes_count=01，说明方法表后面有 1 个属性。

2.9.2、Code 属性表

⑴、 看看紧接着在索引 0x2193 的 2 个字节(0x0160=352["Code"])，说明后面是一个 Code 属性，接下来在索引 0x2195 的 4 个字节(0x5C=92)，表示 code 属性除了前面 2 个字段的 6 个字节，还有 92 个字节(从索引 0x2199 开始)。
max_stack=03，该方法有包括"this"在内的 4 个操作数，操作数栈不会超过 3 这个深度。
max_locals=04，该方法有包括"this"在内的 4 个操作数。
⑵、4 字节的 code_length=0x0c=12，即后面有 12 个字节码指令：
2A 2C 2D B7 01 49 2A 2B B5 01 2F B1
⑶、其中
0x2A　aload_0 将第 1 个引用类型本地变量推送至栈顶
0x2C　　aload_2 将第 3 个引用类型本地变量推送至栈顶
0x2D　aload_3 将第 4 个引用类型本地变量推送至栈顶
注：aload_0 在非静态方法中,表示对 this 的操作，在 static 方法中,表示对方法的第 1 个参数的操作。接下来的是 3 个引用型参数。
0xB7　invokespecial 这条指令是以栈顶的引用类型(即第 1 个参数 this)所指向的对象作为方法接收者，调用对对象的父类构造方法，实例初始化方法，私有方法。
0x0149=329　这两个字节，是常量池的方法常量的索引，是 invokespecial 指令要执行的方法

即前面 6 个字节的代码可翻译为
this.super(assistant, part);
⑷、再接下来的 0x2A 与 0x2B 指令是将第 1 与第 2 个引用类型本地变量推送至栈顶
0xB5　　putfield　为指定类实例的域赋值。
0x012F=303　这两个字节，是常量池的字段常量的索引，是 putfield 指令要赋值字段。



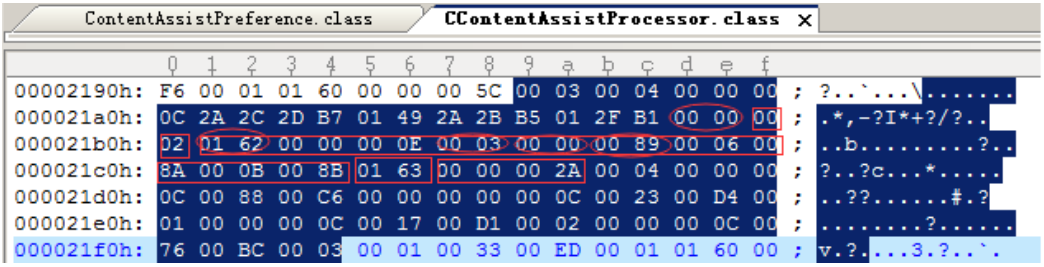
即前面 6 个字节的代码可翻译为
this. fEditor= editor;

⑸、0xB1　　return　　从当前方法返回 void。

⑹、根据上面的分析，这段代码可以表示为：

```
public CContentAssistProcessor(IEditorPart editor, ContentAssistant assistant, String part) {
    super(assistant, partition);
    fEditor= editor;
}
```

⑺接下来的 exception_table_length=0(在索引 0x21ad 的 2 个字节)表示该方法没有异常表。



⑻、接下来的是 Code 属性表中嵌套的 2 个属性表。

先来分析第一个属性

①、 前面 2 字节(索引 0x21b1)为 0x0162=354,指向常量池的 String 常量" LineNumberTable ".

LineNumberTable 属性结构

类　型	名　称	数　量
u2	attribute_name_index	1
u4	attribute_length	1
u2	line_number_table_length	1
line_number_info	line_number_table	line_number_table_length

②、 接下来的 4 字节 0x0000000E=14 是行号属性表的长度，即后面有 14 个字节。

③、 紧随其后的 2 字节((在索引 0x21b7 处) 为 0x0003，表示 line_number_info 的数目， line_number_info 表示字节码与 java 源码对应关系，是两个 u2 类型的数据项，第一个 u2 表示字节码字节索引，第二个 u2 表示 java 源码行号。即 2 + 4 *3 =14 个字节，刚好与上面吻合。

	字节码字节索引	java 源码行号	
A、	0x00	0x89=137	
B、	0x06	0x8A=138	
C、	0x0B	0x8B=139	(即 B1 指令的相对位置)

与源代码文件吻合

```
136  public CContentAssistProcessor(IEditorPart editor, ContentAssistant assistant, String partition) {
137      super(assistant, partition);
138      fEditor= editor;
139  }
```

再来分析第二个属性

Ⅰ、前面 2 字节(索引 0x21c5)为 0x0163=355,指向常量池的 String 常量" LocalVariableTable ".

局部变量表 LocalVariableTable 的结构如下图所示

LocalVariableTable 属性结构

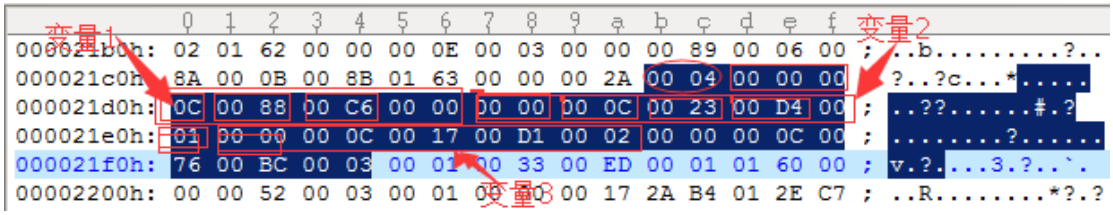
类　型	名　称	数　量
u2	attribute_name_index	1
u4	attribute_length	1
u2	local_variable_table_length	1
local_variable_info	local_variable_table	local_variable_table_length

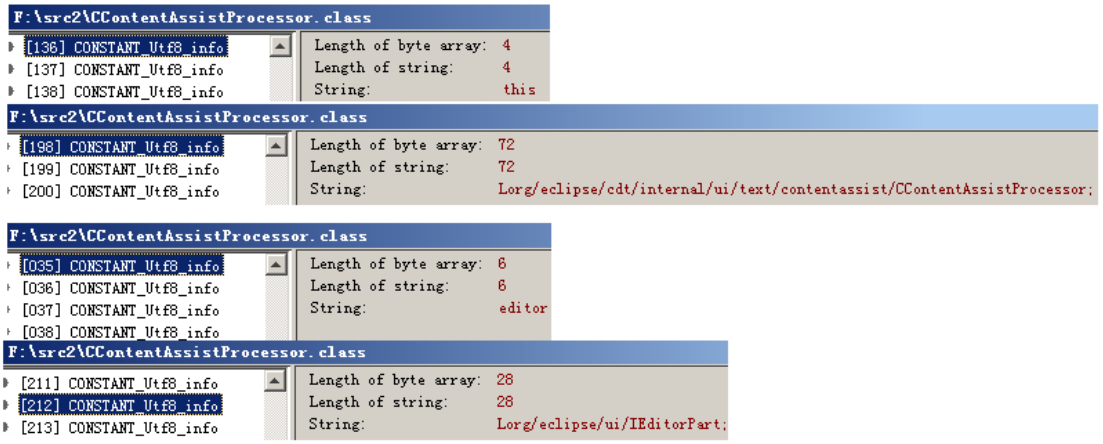
local_variable_info 项目结构

类　型	名　称	数　量
u2	start_pc	1
u2	length	1
u2	name_index	1
u2	descriptor_index	1
u2	index	1

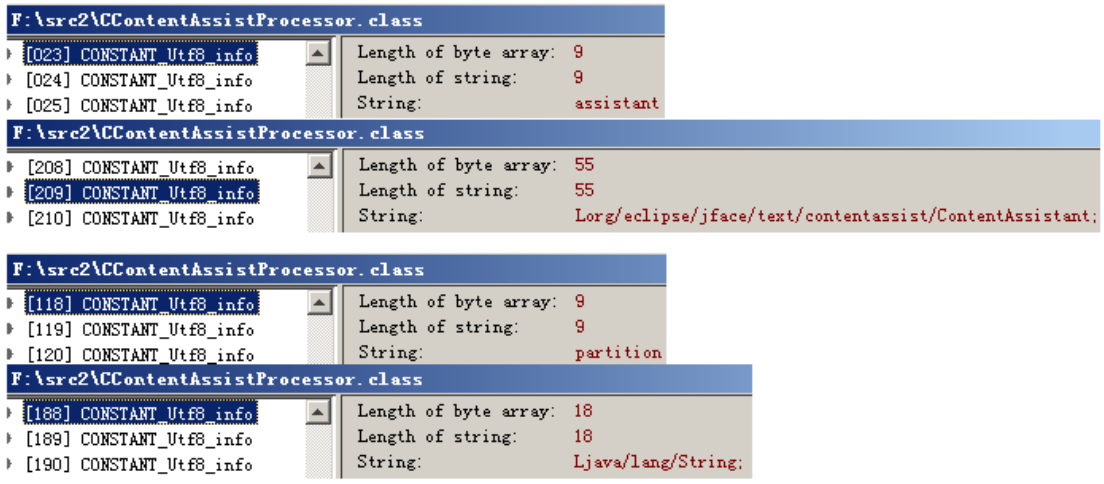
Ⅱ、紧接着的 4 字节 0x0000002A=42，表示局部变量表属性除了前面 2 个字段的 6 个字节，还有 42 个字节(从索引 0x21cb 开始)。
Ⅲ、下 1 个 u2 类型的数据为 0x04 表示有 4 个局部变量信息，与上面的分析一致。
字节长度共 2 + 10*4 ＝42 字节，与上面相符。
start_pc 与 length 属性分别代表了这个局部变量作用域开始字节码偏移量与范围长度。
name_index 与 descriptor_index 指向常量池的字符串常量，表示局部变量名称及其描述符。
index 是这个局部变量在变量表中的位置。

Ⅳ、变量	start_pc	length	name_index	descriptor_index	index
1	0x00	0x0C	0x0088=136	0x00C6=198	0
2	0x00	0x0C	0x0023=35	0x00D4=212	1
3	0x00	0x0C	0x0017=23	0x00D1=209	2
4	0x00	0x0C	0x0076=118	0x00BC=188	3





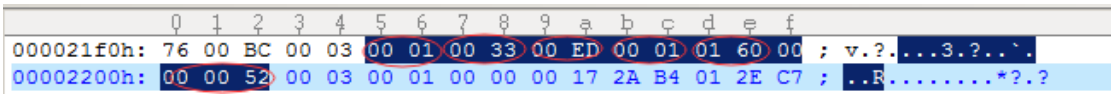
第 1 与第 2 个变量 this 与 editor;



第 3 与第 4 个变量 assistant 与 partition。
综上所述，Code 属性的代码信息从索引 0x2199 开始，到 0x 21F4 结束，刚好 92 个字节。

2.10、分析第 2-11 个方法

2.10.1、第 2 个方法从字节索引 0x21F5 开始



①、方法表属性

access_flag=1,public;　name_index=0x33=51["<getContextInformationValidator>"];

descriptor_index=0xED=237

["()Lorg/eclipse/jface/text/contentassist/IContextInformationValidator"];

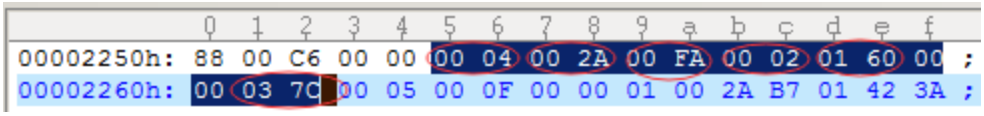
即方法头应为：

public IContextInformationValidator getContextInformationValidator() ;

②、attributes_count=01，说明方法表后面有 1 个属性。

③、紧接着在索引 0x21FD 的 2 个字节(0x0160=352["Code"])，说明后面是一个 Code 属性，接下来在索引 0x21FF 的 4 个字节(0x52=82)，表示 code 属性除了前面 2 个字段的 6 个字节，还有 82 个字节(从索引 0x2203 开始)。

2. 10.2、第 3 个方法从字节索引 0x2255 开始



①、方法表属性

access_flag=4, protected;　name_index=0x2A=42["<filterAndSortProposals>"];

descriptor_index=0xFA=250

["(Ljava/util/List;　Lorg/eclipse/core/runtime/ IProgressMonitor;

　　Lorg/eclipse/cdt/ui/text/contentassist/ContentAssistInvocationContext;) Ljava/util/List"];

即方法头应为：

public List<? > filterAndSortProposals(List<?> proposals,

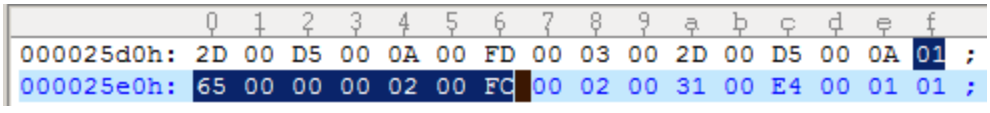
　　IProgressMonitor monitor, ContentAssistInvocationContext context) ;

②、attributes_count=02，说明方法表后面有 2 个属性。

③、紧接着在索引 0x225D 的 2 个字节(0x0160=352["Code"])，说明后面是一个 Code 属性，接下来在索引 0x225F 的 4 个字节(0x037C=892)，表示 code 属性除了前面 2 个字段的 6 个字节，还有 892 个字节(从索引 0x2263 开始)。

即 Code 属性最后一个字节索引为：0x2262 + 0x37C = 0x25DE。

④、从 0x25DF 开始是第 2 个属性的数据。

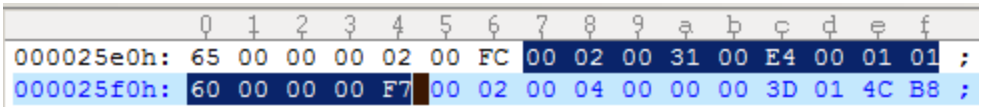


0x25DF 开始的 2 个字节(0x0165=357["Signature "])，说明后面是一个 Signature 属性，结构为：

```
Signature_attribute {
        u2 attribute_name_index;
        u4 attribute_length;
        u2 signature_index;
}
```

⑤、Signature 属性是泛型方法的特征，该属性长度为 2 字节，最后 2 字节 0xFC=252 指向字符串常量，其值为：
(Ljava/util/List<Lorg/eclipse/jface/text/contentassist/ICompletionProposal;>;Lorg/eclipse/core/runtime/IProgressMonitor;Lorg/eclipse/cdt/ui/text/contentassist/ContentAssistInvocationContext;)Ljava/util/List<Lorg/eclipse/jface/text/contentassist/ICompletionProposal;>;
⑥综上所述，第 3 个方法应为：
protected List<ICompletionProposal> filterAndSortProposals(List<ICompletionProposal> proposals, IProgressMonitor monitor, ContentAssistInvocationContext context)

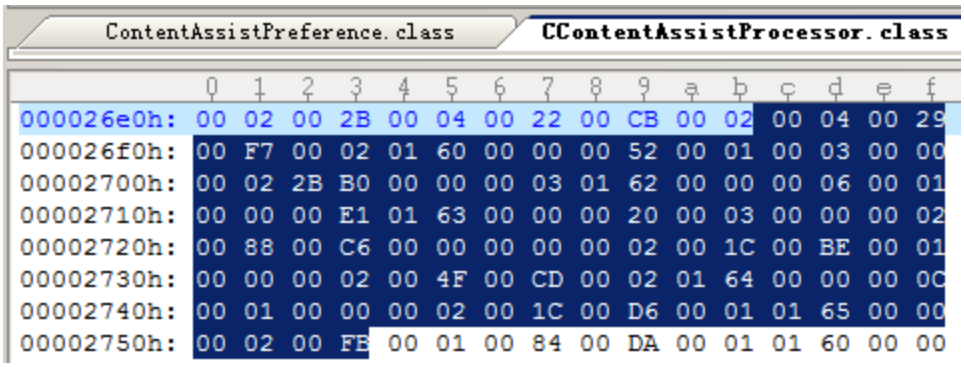
2.10.3、第 4 个方法从字节索引 0x25E7 开始



①、方法表属性
access_flag=2, private; name_index=0x31=49["<getCompletionFilter>"];
descriptor_index=0xE4=228
["()Lorg/eclipse/cdt/ui/text/contentassist/IProposalFilter ;"];
即方法头应为：
private IProposalFilter getCompletionFilter();

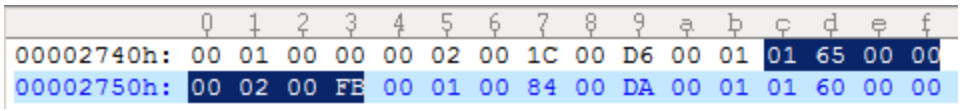
②、attributes_count=01，说明方法表后面有 1 个属性。
③、紧接着在索引 0x25EF 的 2 个字节(0x0160=352["Code"])，说明后面是一个 Code 属性，接下来在索引 0x25F1 的 4 个字节(0xF7=247)，表示 code 属性除了前面 2 个字段的 6 个字节，还有 247 个字节(从索引 0x25F5 开始)。
即第 4 个方法属性结束的字节索引为 0x25F4 + 0xF7 = 0x26EB。

2.10.4、第 5 个方法从字节索引 0x26EC 开始



①、方法表属性
access_flag=4, protected; name_index=0x29=41["<filterAndSortContextInformation >"];
descriptor_index=0xF7=247
["(Ljava/util/List; Lorg/eclipse/core/runtime/ IProgressMonitor;) Ljava/util/List"];;
即方法头应为：
protected List<? > filterAndSortContextInformation(List<? > contexts, IProgressMonitor monitor)
②、attributes_count=02，说明方法表后面有 2 个属性。

③、紧接着在索引 0x26F4 的 2 个字节(0x0160=352["Code"]), 说明后面是一个 Code 属性，接下来在索引 0x26F6 的 4 个字节(0x52=82), 表示 code 属性除了前面 2 个字段的 6 个字节，还有 82 个字节(从索引 0x26FA 开始)。
即 Code 属性最后一个字节索引为：0x26F9 + 0x52 = 0x274B。
④、从 0x274C 开始是第 2 个属性的数据。

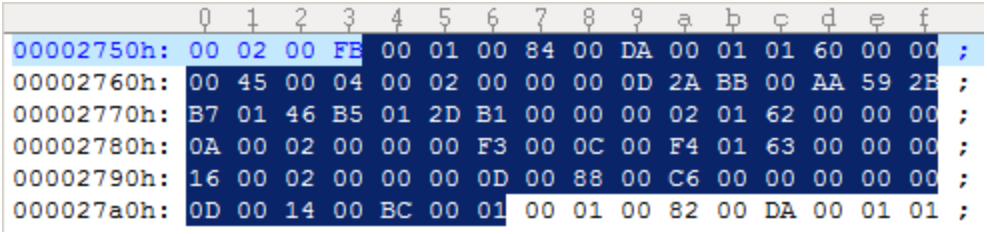


0x274C 开始的 2 个字节(0x0165=357["Signature "]), 说明后面是一个 Signature 属性，结构为：

```
Signature_attribute {
        u2 attribute_name_index;
        u4 attribute_length;
        u2 signature_index;
}
```

⑤、Signature 属性是泛型方法的特征，该属性长度为 2 字节，最后 2 字节 0xFB=251 指向字符串常量，其值为：
(Ljava/util/List<Lorg/eclipse/jface/text/contentassist/IContextInformation;>;
Lorg/eclipse/core/runtime/IProgressMonitor;)Ljava/util/List<Lorg/eclipse/jface/text/contentassist/ IContextInformation;>;
⑥综上所述，第 5 个方法应为：
protected List<IContextInformation> filterAndSortContextInformation(List<IContextInformation> contexts, IProgressMonitor monitor);

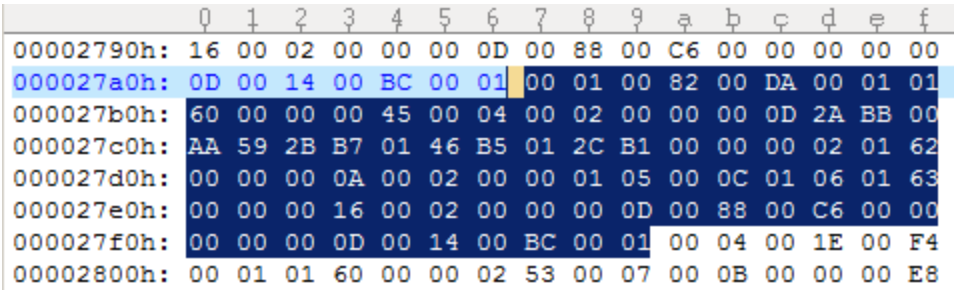
2. 10.5、第 6 个方法从字节索引 0x2754 开始



①、方法表属性
access_flag=1, public; name_index=0x84=132["<setReplacementAutoActivationCharacters>"];
descriptor_index=0xDA=218
["(Ljava/lang/String;)V"];
即方法头应为：
public void setReplacementAutoActivationCharacters(String activationSet);

②、attributes_count=01, 说明方法表后面有 1 个属性。
③、紧接着在索引 0x275C 的 2 个字节(0x0160=352["Code"]), 说明后面是一个 Code 属性，接下来在索引 0x275E 的 4 个字节(0x45=69), 表示 code 属性除了前面 2 个字段的 6 个字节，还有 69 个字节(从索引 0x2762 开始)。
即第 6 个方法属性结束的字节索引为 0x2761 + 0x45 = 0x27A6。

2. 10.6、第 7 个方法从字节索引 0x27A7 开始



	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00002790h:	16	00	02	00	00	00	0D	00	88	00	C6	00	00	00	00	00
000027a0h:	0D	00	14	00	BC	00	01	00	01	00	82	00	DA	00	01	01
000027b0h:	60	00	00	00	45	00	04	00	02	00	00	00	0D	2A	BB	00
000027c0h:	AA	59	2B	B7	01	46	B5	01	2C	B1	00	00	00	02	01	62
000027d0h:	00	00	00	0A	00	02	00	00	01	05	00	0C	01	06	01	63
000027e0h:	00	00	00	16	00	02	00	00	00	0D	00	88	00	C6	00	00
000027f0h:	00	00	00	0D	00	14	00	BC	00	01	00	04	00	1E	00	F4
00002800h:	00	01	01	60	00	00	02	53	00	07	00	0B	00	00	00	E8

①、方法表属性

access_flag=1, public; name_index=0x82=130["<setCContentAutoActivationCharacters>"];

descriptor_index=0xDA=218

["(Ljava/lang/String;)V"];

即方法头应为：

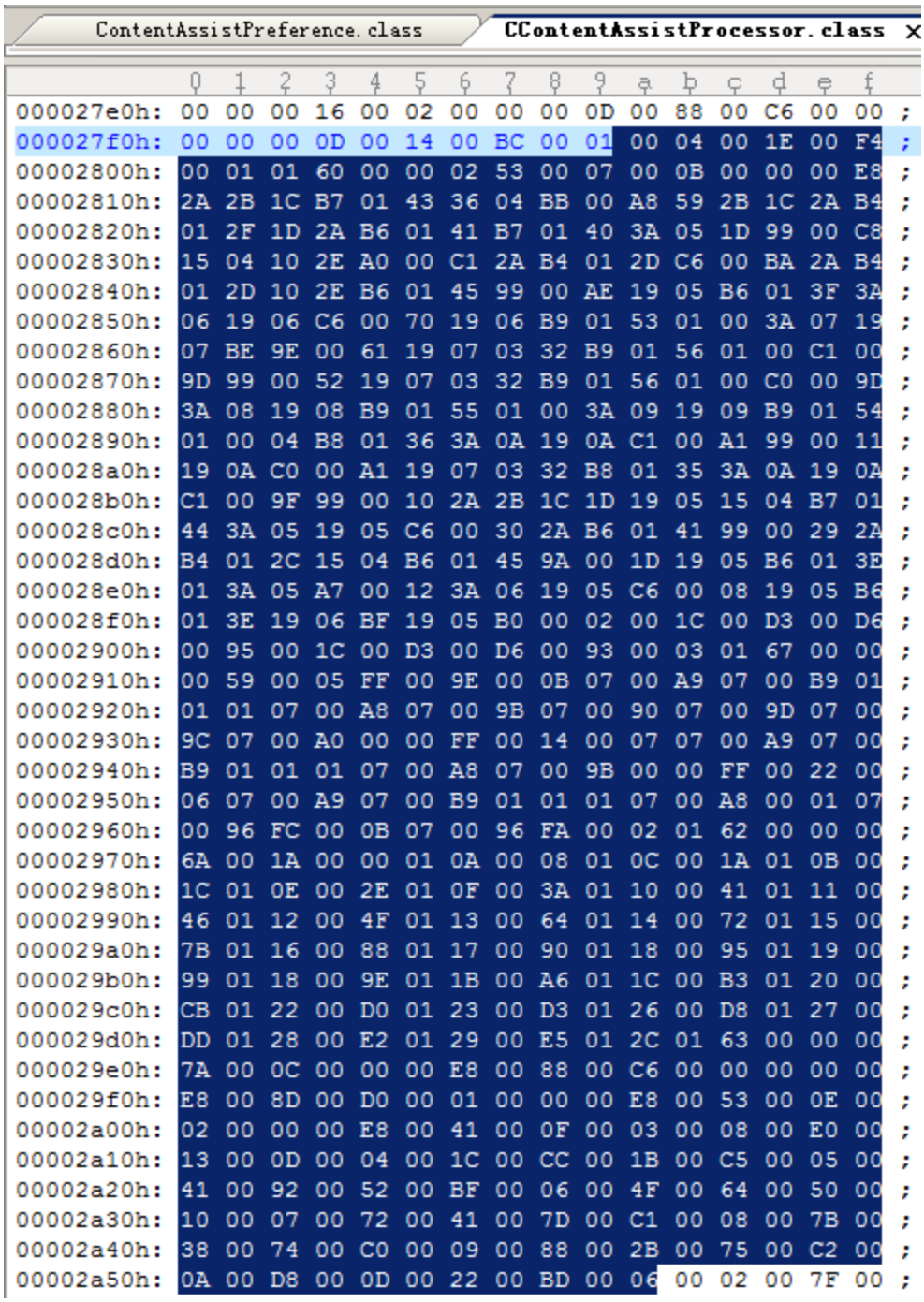
public void setCContentAutoActivationCharacters(String activationSet);

②、attributes_count=01，说明方法表后面有 1 个属性。

③、紧接着在索引 0x27AF 的 2 个字节(0x0160=352["Code"])，说明后面是一个 Code 属性，接下来在索引 0x27B1 的 4 个字节(0x45=69)，表示 code 属性除了前面 2 个字段的 6 个字节，还有 69 个字节(从索引 0x27B5 开始)。

即第 7 个方法属性结束的字节索引为 0x27B4 + 0x45 = 0x27F9。

2. 10.7、第 8 个方法从字节索引 0x27FA 开始



①、方法表属性

access_flag=4, protected;　name_index=0x1E=30["<createContext>"];

descriptor_index=0xF4=244

["(Lorg/eclipse/Jface/text/ ITextViewer;IZ)

Lorg/eclipse/cdt/ui/text/contentassist/ContentAssistInvocationContext; "];

即方法头应为：

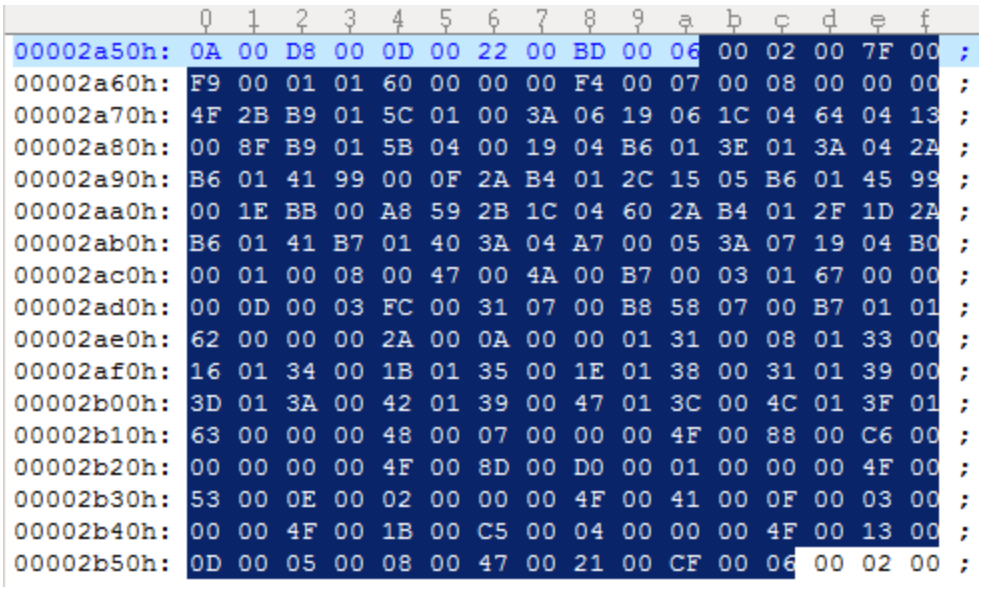
protected ContentAssistInvocationContext createContext(ITextViewer viewer, int offset, boolean isCompletion);

②、attributes_count=01，说明方法表后面有 1 个属性。

③、紧接着在索引 0x2802 的 2 个字节(0x0160=352["Code"])，说明后面是一个 Code 属性，接下来在索引 0x2804 的 4 个字节(0x253=595)，表示 code 属性除了前面 2 个字段的 6 个字节，还有 595 个字节(从索引 0x2808 开始)。

即第 8 个方法属性结束的字节索引为 0x2807 + 0x253 = 0x2A5A。

2. 10.8、第 9 个方法从字节索引 0x2A5B 开始



```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00002a50h: 0A 00 D8 00 0D 00 22 00 BD 00 06 00 02 00 7F 00 ;
00002a60h: F9 00 01 01 60 00 00 00 F4 00 07 00 08 00 00 00 ;
00002a70h: 4F 2B B9 01 5C 01 00 3A 06 19 06 1C 04 64 04 13 ;
00002a80h: 00 8F B9 01 5B 04 00 19 04 B6 01 3E 01 3A 04 2A ;
00002a90h: B6 01 41 99 00 0F 2A B4 01 2C 15 05 B6 01 45 99 ;
00002aa0h: 00 1E BB 00 A8 59 2B 1C 04 60 2A B4 01 2F 1D 2A ;
00002ab0h: B6 01 41 B7 01 40 3A 04 A7 00 05 3A 07 19 04 B0 ;
00002ac0h: 00 01 00 08 00 47 00 4A 00 B7 00 03 01 67 00 00 ;
00002ad0h: 00 0D 00 03 FC 00 31 07 00 B8 58 07 00 B7 01 01 ;
00002ae0h: 62 00 00 00 2A 00 0A 00 00 01 31 00 08 01 33 00 ;
00002af0h: 16 01 34 00 1B 01 35 00 1E 01 38 00 31 01 39 00 ;
00002b00h: 3D 01 3A 00 42 01 39 00 47 01 3C 00 4C 01 3F 01 ;
00002b10h: 63 00 00 00 48 00 07 00 00 00 4F 00 88 00 C6 00 ;
00002b20h: 00 00 00 00 4F 00 8D 00 D0 00 01 00 00 00 4F 00 ;
00002b30h: 53 00 0E 00 02 00 00 00 4F 00 41 00 0F 00 03 00 ;
00002b40h: 00 00 4F 00 1B 00 C5 00 04 00 00 00 4F 00 13 00 ;
00002b50h: 0D 00 05 00 08 00 47 00 21 00 CF 00 06 00 02 00 ;
```

①、方法表属性

access_flag=2, private; name_index=0x7F=127 ["<replaceDotWithArrow>"];

descriptor_index=0xF9=249

["(Lorg/eclipse/Jface/text/ ITextViewer;IZ

Lorg/eclipse/cdt/internal/ui/text/contentassist/CContentAssistInvocationContext;C)

Lorg/eclipse/cdt/internal/ui/text/contentassist/CContentAssistInvocationContext;"];

即方法头应为：

private CContentAssistInvocationContext replaceDotWithArrow(ITextViewer viewer, int offset,

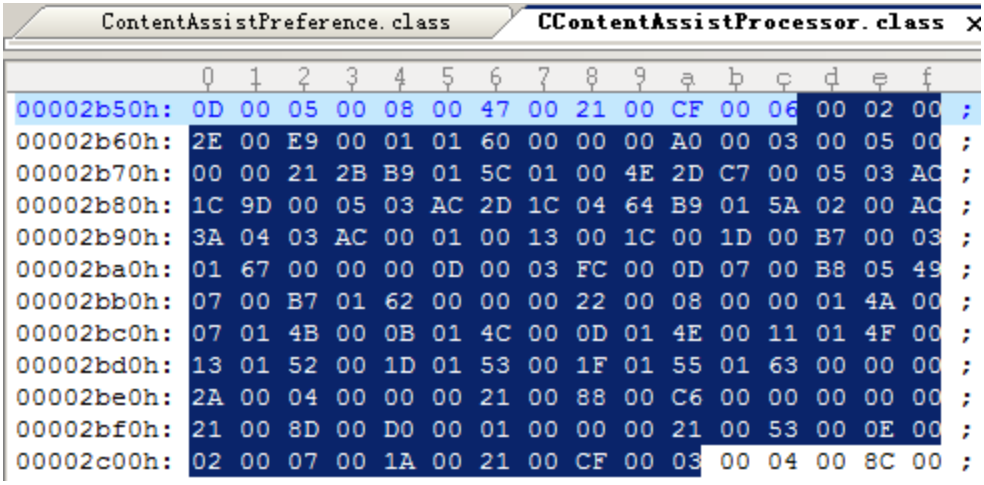
boolean isCompletion, CContentAssistInvocationContext context, char activationChar)

②、attributes_count=01，说明方法表后面有 1 个属性。

③、紧接着在索引 0x2A63 的 2 个字节(0x0160=352["Code"])，说明后面是一个 Code 属性，接下来在索引 0x2A65 的 4 个字节(0xF4=244)，表示 code 属性除了前面 2 个字段的 6 个字节，还有 244 个字节(从索引 0x2A69 开始)。

即第 9 个方法属性结束的字节索引为 0x2A68 + 0xF4 = 0x2B5C。

2. 10.9、第 10 个方法从字节索引 0x2B5D 开始



①、方法表属性

access_flag=2, private;　name_index=0x2E=46 ["<getActivationChar >"];

descriptor_index=0xE9=233

["(Lorg/eclipse/Jface/text/ ITextViewer;I) C"];

即方法头应为：

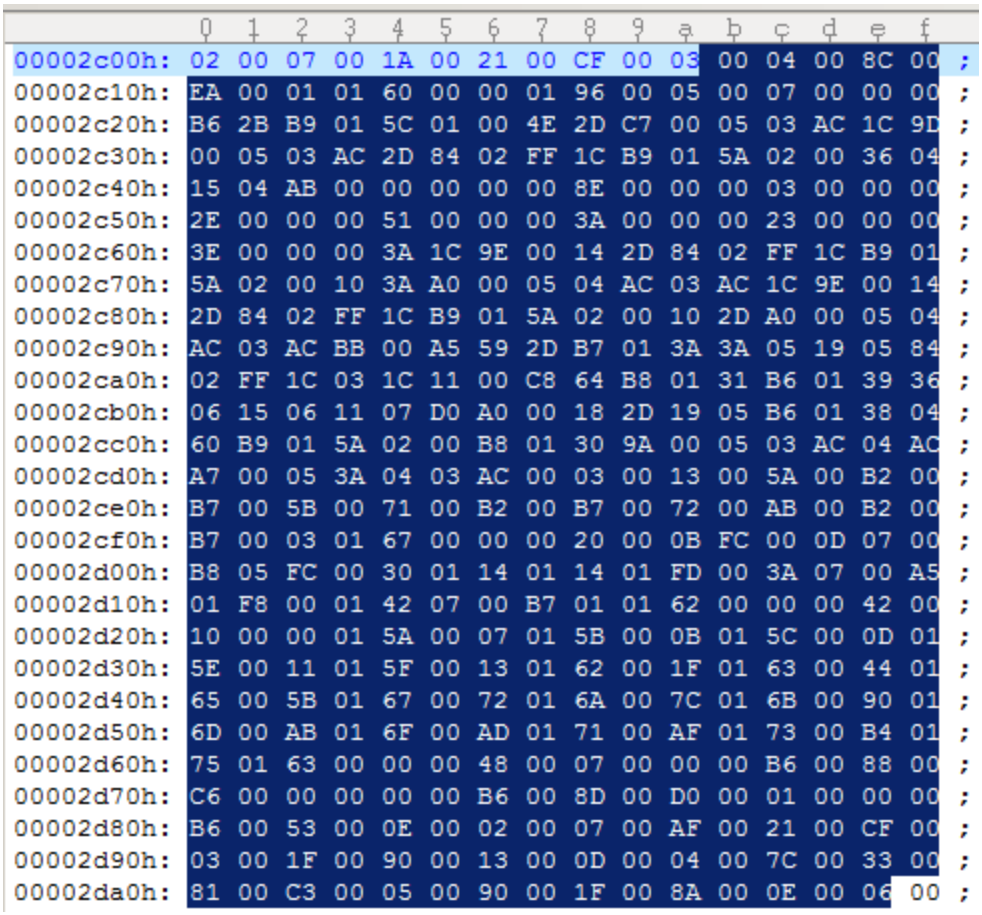
private char getActivationChar(ITextViewer viewer, int offset)

②、attributes_count=01，说明方法表后面有 1 个属性。

③、紧接着在索引 0x2B65 的 2 个字节(0x0160=352["Code"])，说明后面是一个 Code 属性，接下来在索引 0x2B67 的 4 个字节(0xA0=160)，表示 code 属性除了前面 2 个字段的 6 个字节，还有 160 个字节(从索引 0x2B6B 开始)。

即第 10 个方法属性结束的字节索引为 0x2B6A + 0xA0 = 0x2C0A。

2. 10.10、第 11 个方法从字节索引 0x2C0B 开始



```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00002c00h: 02 00 07 00 1A 00 21 00 CF 00 03 00 04 00 8C 00 ;
00002c10h: EA 00 01 01 60 00 00 01 96 00 05 00 07 00 00 00 ;
00002c20h: B6 2B B9 01 5C 01 00 4E 2D C7 00 05 03 AC 1C 9D ;
00002c30h: 00 05 03 AC 2D 84 02 FF 1C B9 01 5A 02 00 36 04 ;
00002c40h: 15 04 AB 00 00 00 00 00 8E 00 00 00 03 00 00 00 ;
00002c50h: 2E 00 00 00 51 00 00 00 3A 00 00 00 23 00 00 00 ;
00002c60h: 3E 00 00 00 3A 1C 9E 00 14 2D 84 02 FF 1C B9 01 ;
00002c70h: 5A 02 00 10 3A A0 00 05 04 AC 03 AC 1C 9E 00 14 ;
00002c80h: 2D 84 02 FF 1C B9 01 5A 02 00 10 2D A0 00 05 04 ;
00002c90h: AC 03 AC BB 00 A5 59 2D B7 01 3A 3A 05 19 05 84 ;
00002ca0h: 02 FF 1C 03 1C 11 00 C8 64 B8 01 31 B6 01 39 36 ;
00002cb0h: 06 15 06 11 07 D0 A0 00 18 2D 19 05 B6 01 38 04 ;
00002cc0h: 60 B9 01 5A 02 00 B8 01 30 9A 00 05 03 AC 04 AC ;
00002cd0h: A7 00 05 3A 04 03 AC 00 03 00 13 00 5A 00 B2 00 ;
00002ce0h: B7 00 5B 00 71 00 B2 00 B7 00 72 00 AB 00 B2 00 ;
00002cf0h: B7 00 03 01 67 00 00 00 20 00 0B FC 00 0D 07 00 ;
00002d00h: B8 05 FC 00 30 01 14 01 14 01 FD 00 3A 07 00 A5 ;
00002d10h: 01 F8 00 01 42 07 00 B7 01 01 62 00 00 00 42 00 ;
00002d20h: 10 00 00 01 5A 00 07 01 5B 00 0B 01 5C 00 0D 01 ;
00002d30h: 5E 00 11 01 5F 00 13 01 62 00 1F 01 63 00 44 01 ;
00002d40h: 65 00 5B 01 67 00 72 01 6A 00 7C 01 6B 00 90 01 ;
00002d50h: 6D 00 AB 01 6F 00 AD 01 71 00 AF 01 73 00 B4 01 ;
00002d60h: 75 01 63 00 00 00 48 00 07 00 00 00 B6 00 88 00 ;
00002d70h: C6 00 00 00 00 00 B6 00 8D 00 D0 00 01 00 00 00 ;
00002d80h: B6 00 53 00 0E 00 02 00 07 00 AF 00 21 00 CF 00 ;
00002d90h: 03 00 1F 00 90 00 13 00 0D 00 04 00 7C 00 33 00 ;
00002da0h: 81 00 C3 00 05 00 90 00 1F 00 8A 00 0E 00 06 00 ;
```

①、方法表属性

access_flag=4, protected;　name_index=0x8C=140 ["<verifyAutoActivation >"];

descriptor_index=0xEA=234

["(Lorg/eclipse/Jface/text/ ITextViewer;I) Z"];

即方法头应为：

protected boolean verifyAutoActivation(ITextViewer viewer, int offset)

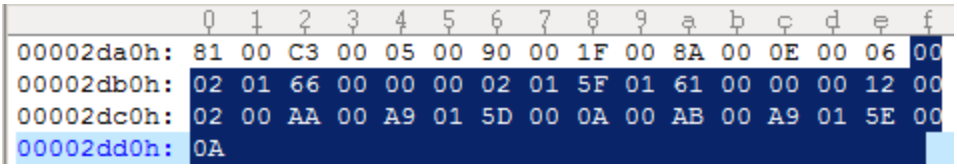
②、attributes_count=01，说明方法表后面有 1 个属性。

③、紧接着在索引 0x2C13 的 2 个字节(0x0160=352["Code"])，说明后面是一个 Code 属性，接下来在索引 0x2C15 的 4 个字节(0x196=406)，表示 code 属性除了前面 2 个字段的 6 个字节，还有 406 个字节(从索引 0x2C19 开始)。接下来的 4 个字节表示 max_stack 与 max_locals 各为 5 与 7。之后索引 0x2C1D 的 4 个字节(0xB6=182)表示有 182 个指令码，最后一个指令码(0xAC ireturn)的字节索引为 0x2C20 + 0xB6 = 0x2CD6。

即第 11 个方法属性结束的字节索引为 0x2C18 + 0x196= 0x2DAE。

2.11、属性表

2.10.1、接下来的 2 个字节(index: 0x2DAF)attributes_count = 2，表示最后有 2 个属性。



```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00002da0h: 81 00 C3 00 05 00 90 00 1F 00 8A 00 0E 00 06 00
00002db0h: 02 01 66 00 00 00 02 01 5F 01 61 00 00 00 12 00
00002dc0h: 02 00 AA 00 A9 01 5D 00 0A 00 AB 00 A9 01 5E 00
00002dd0h: 0A
```

2.10.2、分析第 1 个属性

①、在索引 0x2DB1 的 2 个字节 (0x0166=358["SourceFile"])，说明后面是一个 SourceFile 属性

SourceFile 属性结构

类 型	名 称	数 量
u2	attribute_name_index	1
u4	attribute_length	1
u2	sourcefile_index	1

②、接下来在索引 0x2DB3 的 4 个字节(0x2=2)，是 SourceFile 属性后面字节的长度。

③、跟随其后索引 0x2DB7 的 2 个字节(0x015F=351["CContentAssistProcessor.java"])，是 SourceFile 文件名的字符串常量索引。

2.10.3、分析第 2 个属性

④、在索引 0x2DB9 的 2 个字节 (0x0161=353["InnerClasses"])，说明后面是一个 InnerClasses 属性

InnerClasses 属性结构

类 型	名 称	数 量
u2	attribute_name_index	1
u4	attribute_length	1
u2	number_of_classes	1
inner_classes_info	inner_classes	number_of_classes

inner_classes_info 表的结构

类 型	名 称	数 量
u2	inner_class_info_index	1
u2	outer_class_info_index	1
u2	inner_name_index	1
u2	inner_class_access_flags	1

inner_class_access_flags 标志

标志名称	标志值	含 义
ACC_PUBLIC	0x0001	内部类是否为 public
ACC_PRIVATE	0x0002	内部类是否为 private
ACC_PROTECTED	0x0004	内部类是否为 protected
ACC_STATIC	0x0008	内部类是否为 static
ACC_FINAL	0x0010	内部类是否为 final
ACC_INTERFACE	0x0020	内部类是否为 synchronized
ACC_ABSTRACT	0x0400	内部类是否为 abstract
ACC_SYNTHETIC	0x1000	内部类是否并非由用户代码产生的
ACC_ANNOTATION	0x2000	内部类是否是一个注解
ACC_ENUM	0x4000	内部类是否是一个枚举

内部类属性中，inner_class_info_index 与 outer_class_info_index 都是指向常量池中 CONSTANT_Class_info 型常量的索引，分别代表了内部类和宿主类的符号引用。

inner_name_index 是指向常量池中 CONSTANT_Utf8_info 型常量的索引，代表内部类的名称，如果是匿名内部类，则为 0。

inner_class_access_flags 是内部类的访问标志，类似于类的 access_flags。取值如上表。

⑤、接下来在索引 0x2DBB 的 4 个字节(0x12=18)，是 InnerClasses 属性后面字节的长度。
⑥、跟随其后索引 0x2DBF 的 2 个字节(0x02])，是内部类的数目。
⑦、第 1 个内部类的值为　00 AA 00 A9 01 5D 00 0A

inner_class_info_index：0xAA=170:

outer_class_info_index：0xA9=169:

inner_name_index:　0x 015D = 349["ActivationSet"]

inner_class_access_flags: 0x0A[private, static]

即内部类可表示为: private static class CContentAssistProcessor. ActivationSet{};

⑧、第 2 个内部类的值为 00 AB 00 A9 01 5E 00 0A

inner_class_info_index：0xAB=171:

outer_class_info_index：0xA9=169:

inner_name_index:　0x 015E = 350["CCompletionProposalWrapper"]

inner_class_access_flags: 0x0A[private, static]

即内部类可表示为: private static class CContentAssistProcessor.
CCompletionProposalWrapper{};