一:源代码

```
package com.tuling.smlz.jvm.classbyatecode;

/**
 * Created by smlz on 2019/11/5.
 */
public class TulingByteCode {

 private String userName;

  public String getUserName() {
  return userName;
  }

  public void setUserName(String userName) {
  this.userName = userName;
  }
}
```

二:通过我们javap -verbose TulingByteCode .class反编译

```
//表示我们通过反编译的来源是哪个字节码文件
Classfile /D:/work_space/idea_space/spring-cloud-source/tuling-jvm/target/classes/com/tuling/smlz/jvm/classbyatecode/TulingByteCode.class
  //最后修改日期；文件大小
  Last modified 2019-11-5; size 629 bytes
  //文件的md5值
  MD5 checksum a0a9c001787f00738627278b0946a388
  //.class文件是通过哪个源文件编译过来的
  Compiled from "TulingByteCode.java"
  //字节码的详细信息
public class com.tuling.smlz.jvm.classbyatecode.TulingByteCode
  //jdk的次版本号
  minor version: 0
  //jdk的主版本号
  major version: 52
  //访问权限
  flags: ACC_PUBLIC, ACC_SUPER
  //常量池
Constant pool:
```

```
  #1 = Methodref #4.#21 // java/lang/Object."<init>":()V
  #2 = Fieldref #3.#22 // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.
userName:Ljava/lang/String;
  #3 = Class #23 // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #4 = Class #24 // java/lang/Object
  #5 = Utf8 userName
  #6 = Utf8 Ljava/lang/String;
  #7 = Utf8 <init>
  #8 = Utf8 ()V
  #9 = Utf8 Code
  #10 = Utf8 LineNumberTable
  #11 = Utf8 LocalVariableTable
  #12 = Utf8 this
  #13 = Utf8 Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8 getUserName
  #15 = Utf8 ()Ljava/lang/String;
  #16 = Utf8 setUserName
  #17 = Utf8 (Ljava/lang/String;)V
  #18 = Utf8 MethodParameters
  #19 = Utf8 SourceFile
  #20 = Utf8 TulingByteCode.java
  #21 = NameAndType #7:#8 // "<init>":()V
  #22 = NameAndType #5:#6 // userName:Ljava/lang/String;
  #23 = Utf8 com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8 java/lang/Object
{
  //构造方法
  public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
  descriptor: ()V
  flags: ACC_PUBLIC
  Code:
  stack=1, locals=1, args_size=1
  0: aload_0
  1: invokespecial #1 // Method java/lang/Object."<init>":()V
  4: return
  LineNumberTable:
  line 6: 0
  LocalVariableTable:
  Start Length Slot Name Signature
  0 5 0 this Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  //get方法
  public java.lang.String getUserName();
```

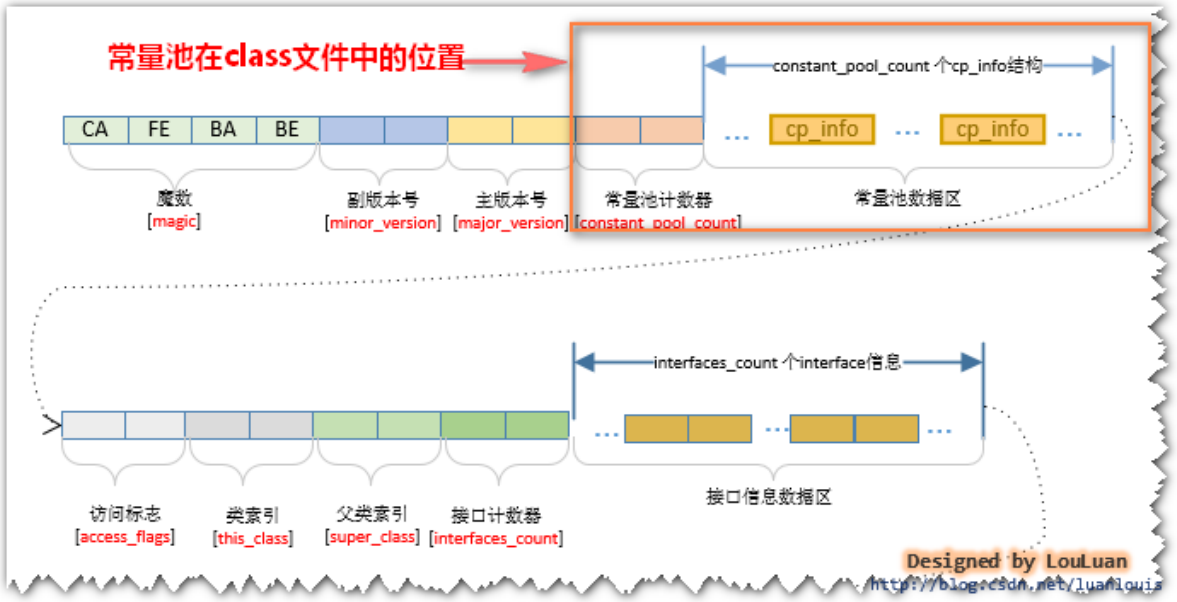
```
  descriptor: ()Ljava/lang/String;
  flags: ACC_PUBLIC
  Code:
  stack=1, locals=1, args_size=1
  0: aload_0
  1: getfield #2 // Field userName:Ljava/lang/String;
  4: areturn
  LineNumberTable:
  line 11: 0
  LocalVariableTable:
  Start Length Slot Name Signature
  0 5 0 this Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  //set方法
  public void setUserName(java.lang.String);
  descriptor: (Ljava/lang/String;)V
  flags: ACC_PUBLIC
  Code:
  stack=2, locals=2, args_size=2
  0: aload_0
  1: aload_1
  2: putfield #2 // Field userName:Ljava/lang/String;
  5: return
  LineNumberTable:
  line 15: 0
  line 16: 5
  LocalVariableTable:
  Start Length Slot Name Signature
  0 6 0 this Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  0 6 1 userName Ljava/lang/String;
  MethodParameters:
  Name Flags
  userName
}
SourceFile: "TulingByteCode.java"

```

三:class文件通过16进制查看器打开如下

通过16进制查看器打开的文件结构是一个当个字节来显示，因为一个16进制数可以通过4位来表示,一个字节8位可以表示二个16进制数

```
00000000  CA FE BA BE 00 00 00 34  00 19 0A 00 04 00 15 09  .......4........
00000010  00 03 00 16 07 00 17 07  00 18 01 00 08 75 73 65  .............use
00000020  72 4E 61 6D 65 01 00 12  4C 6A 61 76 61 2F 6C 61  rName...Ljava/la
00000030  6E 67 2F 53 74 72 69 6E  67 3B 01 00 06 3C 69 6E  ng/String;...<in
00000040  69 74 3E 01 00 03 28 29  56 01 00 04 43 6F 64 65  it>...()V...Code
00000050  01 00 0F 4C 69 6E 65 4E  75 6D 62 65 72 54 61 62  ...LineNumberTab
00000060  6C 65 01 00 12 4C 6F 63  61 6C 56 61 72 69 61 62  le...LocalVariab
00000070  6C 65 54 61 62 6C 65 01  00 04 74 68 69 73 01 00  leTable...this..
00000080  33 4C 63 6F 6D 2F 74 75  6C 69 6E 67 2F 73 6D 6C  3Lcom/tuling/sml
00000090  7A 2F 6A 76 6D 2F 63 6C  61 73 73 62 79 61 74 65  z/jvm/classbyate
000000a0  63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/TulingByteC
000000b0  6F 64 65 3B 01 00 0B 67  65 74 55 73 65 72 4E 61  ode;...getUserNa
000000c0  6D 65 01 00 14 28 29 4C  6A 61 76 61 2F 6C 61 6E  me...()Ljava/lan
000000d0  67 2F 53 74 72 69 6E 67  3B 01 00 0B 73 65 74 55  g/String;...setU
000000e0  73 65 72 4E 61 6D 65 01  00 15 28 4C 6A 61 76 61  serName...(Ljava
000000f0  2F 6C 61 6E 67 2F 53 74  72 69 6E 67 3B 29 56 01  /lang/String;)V.
00000100  00 10 4D 65 74 68 6F 64  50 61 72 61 6D 65 74 65  ..MethodParamete
00000110  72 73 01 00 0A 53 6F 75  72 63 65 46 69 6C 65 01  rs...SourceFile.
00000120  00 13 54 75 6C 69 6E 67  42 79 74 65 43 6F 64 65  ..TulingByteCode
00000130  2E 6A 61 76 61 0C 00 07  00 08 0C 00 05 00 06 01  .java...........
00000140  00 31 63 6F 6D 2F 74 75  6C 69 6E 67 2F 73 6D 6C  .1com/tuling/sml
00000150  7A 2F 6A 76 6D 2F 63 6C  61 73 73 62 79 61 74 65  z/jvm/classbyate
00000160  63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/TulingByteC
00000170  6F 64 65 01 00 10 6A 61  76 61 2F 6C 61 6E 67 2F  ode...java/lang/
00000180  4F 62 6A 65 63 74 00 21  00 03 00 04 00 00 00 01  Object.!........
00000190  00 02 00 05 00 06 00 00  00 03 00 01 00 07 00 08  ................
000001a0  00 01 00 09 00 00 00 2F  00 01 00 01 00 00 00 05  ......./........
000001b0  2A B7 00 01 B1 00 00 00  02 00 0A 00 00 00 06 00  *...............
000001c0  01 00 00 00 06 00 0B 00  00 00 0C 00 01 00 00 00  ................
000001d0  05 00 0C 00 0D 00 00 00  01 00 0E 00 0F 00 01 00  ................
000001e0  09 00 00 00 2F 00 01 00  01 00 00 00 05 2A B4 00  ..../........*..
000001f0  02 B0 00 00 00 02 00 0A  00 00 00 06 00 01 00 00  ................
00000200  00 0B 00 0B 00 00 00 0C  00 01 00 00 00 05 00 0C  ................
00000210  00 0D 00 00 00 01 00 10  00 11 00 02 00 09 00 00  ................
00000220  00 3E 00 02 00 02 00 00  00 06 2A 2B B5 00 02 B1  .>........*+....
00000230  00 00 00 02 00 0A 00 00  00 0A 00 02 00 00 00 0F  ................
00000240  00 05 00 10 00 0B 00 00  00 16 00 02 00 00 00 06  ................
00000250  00 0C 00 0D 00 00 00 00  00 06 00 05 00 06 00 01  ................
00000260  00 12 00 00 00 05 01 00  05 00 00 00 01 00 13 00  ................
00000270  00 00 02 00 14                                    .....
```

我们class文件结构图



Class文件结构参照表:

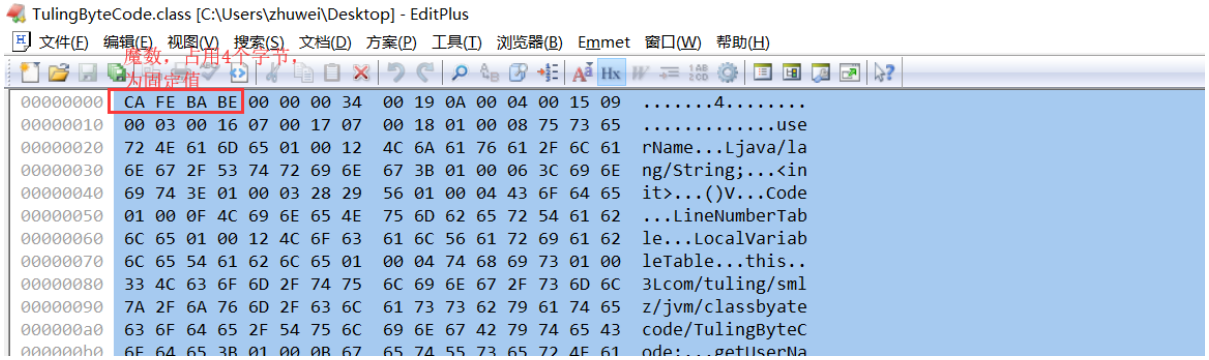
Class文件结构		
类型	名称	数量
u4固定值(0xCAFEBABE)	Magic Number(魔数)	1
u2(2个字节)	次版本号	1
u2(2个字节)	主版本号	1
u2(2个字节)	constant_pool_cout(常量个数)	1
cp_info(N个字节)	constant_pool(常量池表)	constant_pool_cout-1
u2(2个字节)	access_flag(访问标记符号)	1
u2(2个字节)	This class Name	1
U2(2个字节)	super class name	1
u2(2个字节)	Interfaces_count(接口数)	1
u2(二个字节)	interfaces(接口名称)	Interfaces_count
u2(二个字节)	fields_count	1
field_info(n个字节)	fileds (字段表)	fields_count
u2(二个字节)	methods_count(方法个数)	1
method_info(N个字节)	方法表	methods_count
u2(二个字节)	attruibute_count(附加属性个数)	1
attribute_info(n个字节)	attrubites(附加属性表)	attruibute_count

Class文件结构伪代码

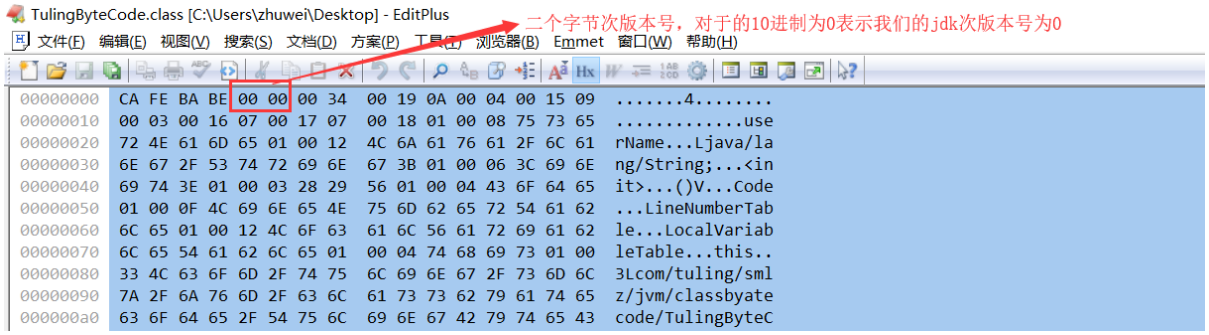
```
ClassFile{
    u2              magicNum;
    u2              minor_version;
    u2              major_version;
    u2              constant_pool_count;
    cp_info         constant_pool[constant_pool_count-1];
    u2              access_flag;
    u2              this_class;
    u2              super_class;
    u2              interfaces_count;
    u2              interfaces[interfaces_count];
    u2              fields_count;
    field_info      fields[fields_count];
    u2              methods_count;
    methods_info    methods[methods_count] ;
    u2              attributes_count;
    atributes_info  attributes[ attributes_count];
}
```

3.1)我们通过javap -verbose来分析一个字节码的时候，将会分析字节码文件的魔数,主 次版本号,常量池，类信息，类的构造方法，类的中的方法信息，类变量与成员变量等信息.

魔数: 文件的开头的 四个字节 是固定 值位 0xCAFEBABE

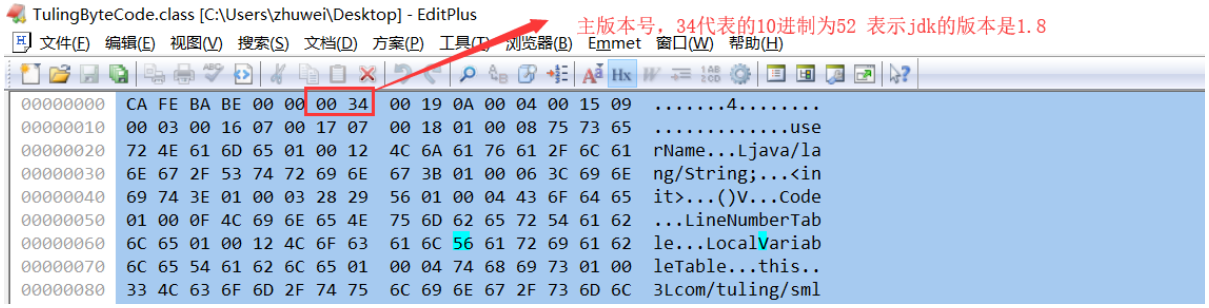


3.2)次版本号(minor version):二个字节00 00 表示jdk的次版本号

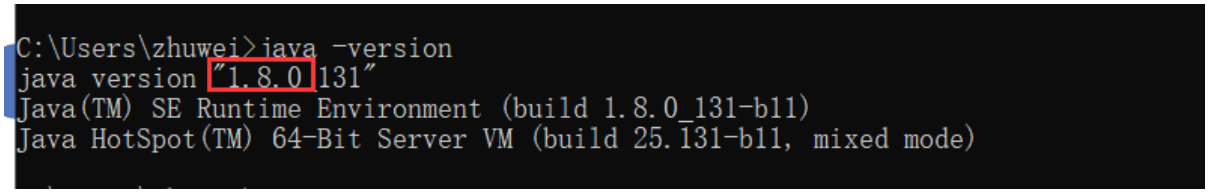


3.3)主版本号(major version):二个字节 00 34 表示为jdk的主版本号，34对于10进制为52

那么52代表的是1.8，51代表的是1.7 等等一直类推下去



所以通过主次版本号来确定我们jdk的版本是1.8.0



3.4)常量池入口，占用二个字节,表示常量池中的个数=00 19 (25)-1=24个，为啥需要-1，因为常量池中的第0个位置被我们的jvm占用了表示为null 所以我们通过编译出来的常量池索引是从1开始的.

```
Constant pool:
  #1 = Methodref #4.#21 // java/lang/Object."<init>":()V
  #2 = Fieldref #3.#22 // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.u
serName:Ljava/lang/String;
  #3 = Class #23 // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #4 = Class #24 // java/lang/Object
  #5 = Utf8 userName
```

```
#6 = Utf8 Ljava/lang/String;
#7 = Utf8 <init>
#8 = Utf8 ()V
 #9 = Utf8 Code
 #10 = Utf8 LineNumberTable
 #11 = Utf8 LocalVariableTable
 #12 = Utf8 this
 #13 = Utf8 Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
 #14 = Utf8 getUserName
 #15 = Utf8 ()Ljava/lang/String;
 #16 = Utf8 setUserName
 #17 = Utf8 (Ljava/lang/String;)V
 #18 = Utf8 MethodParameters
 #19 = Utf8 SourceFile
 #20 = Utf8 TulingByteCode.java
 #21 = NameAndType #7:#8 // "<init>":()V
 #22 = NameAndType #5:#6 // userName:Ljava/lang/String;
 #23 = Utf8 com/tuling/smlz/jvm/classbyatecode/TulingByteCode
 #24 = Utf8 java/lang/Object
```

3.4.1）常量池结构表如图所示

u1,u2,u4,u8分别代表1个字节,2个字节,4个字节,8个字节的无符号数

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

https://blog.csdn.net/qq_39375211

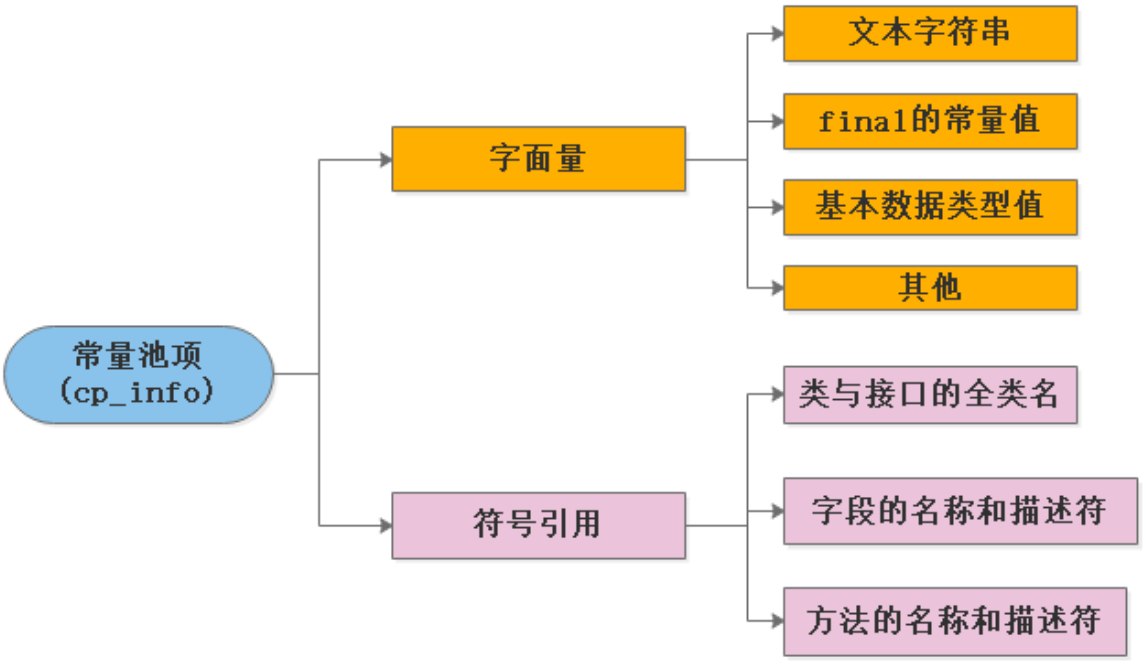
常量	项目	类型	描述
CONSTANT_Methodref_info	tag	u1	值为 10
	index	u2	指向声明方法的类描述符 CONSTANT_Class_info 的索引项
	index	u2	指向名称及类型描述符 CONSTANT_NameAndType 的索引项
CONSTANT_InterfaceMethodref_info	tag	u1	值为 11
	index	u2	指向声明方法的接口描述符 CONSTANT_Class_info 的索引项
	index	u2	指向名称及类型描述符 CONSTANT_NameAndType 的索引项
CONSTANT_NameAndType_info	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

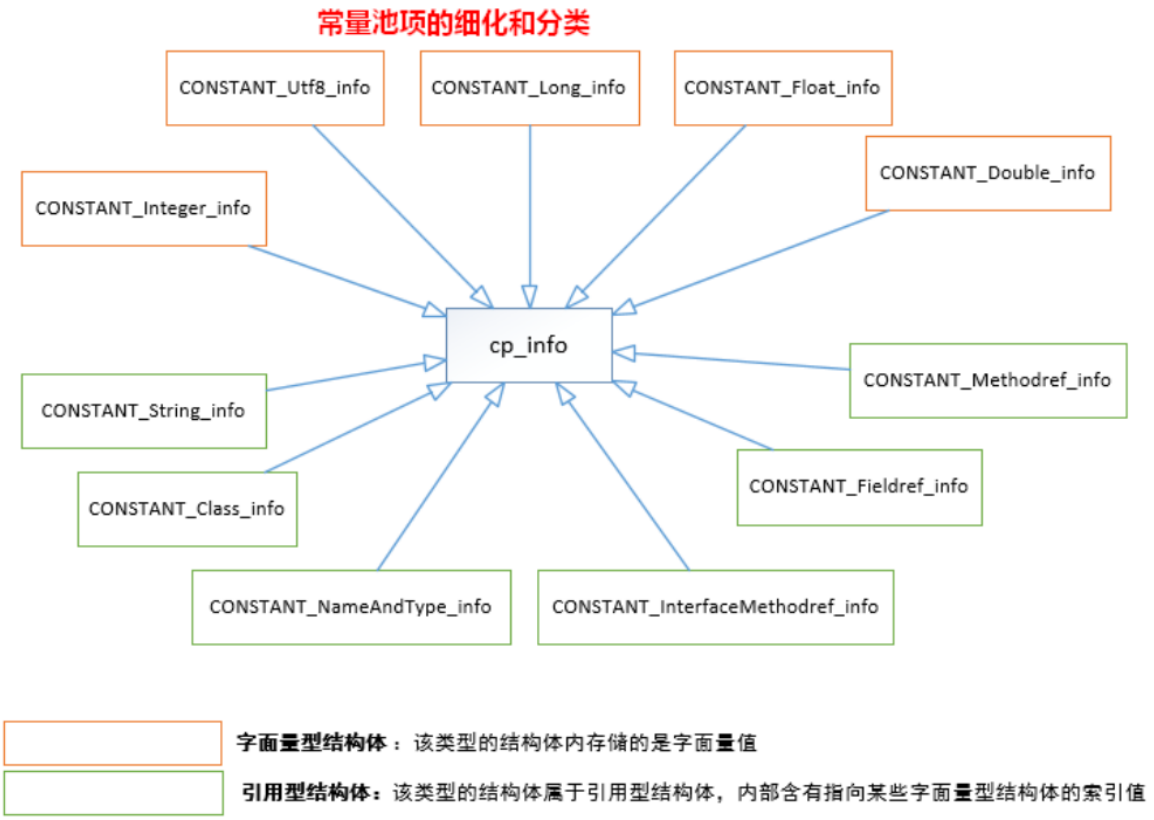
https://blog.csdn.net/qq_39375211

图示:

3.4.2)我们的常量池可以看作我们的java class类的一个资源仓库(比如Java类定的方法和变量信息),我们后面的方法 类的信息的描述信息都是通过索引去常量池中获取。

1)常量池中主要存放二种常量,一种是字面量 一种是符号引用





3.4.3)在JVM规范中，每个字段或者变量都有描述信息,描述信息的主要作用是 数据类型，方法参数列表,返回值类型等.

1)基本参数类型和void类型都是用一个大写的字符来表示，对象类型是通过一个大写L加全类名表示，这么做的好处就是在保证jvm能读懂class文件的情况下尽量的压缩class文件体积.

基本数据类型表示:

B---->byte

C---->char

D---->double

F----->float

I------>int

J------>long

S------>short

Z------>boolean

V------->void

对象类型:

String------>Ljava/lang/String;(后面有一个分号)

对于数组类型: 每一个唯独都是用一个前置 [来表示

比如: int[] ------>[I,

String [][]------>[[Ljava.lang.String;

2）用描述符来描述方法的,先参数列表，后返回值的格式，参数列表按照严格的顺序放在()中

比如源码　String getUserInfoByIdAndName(int id,String name) 的方法描述符号

(I,Ljava/lang/String;）Ljava/lang/String;

第1个常量池分析：0A 00 04 00 15

0A:表示是常量池中的常量类型为方法引用

CONSTANT_Methodref_info 是个比较复杂的常量池项，它的tag值为 10，它内部包含两个常量池索引值。
class_index 指向常量池中的CONSTANT_Class_info 常量池项，该常量池项表示此方法所在的类；
Name_and_type_index指向常量池中的CONSTANT_NameAndType_info常量池项，该常量池项描述这个方法的名称和方法描述符。

00 04二个字节表示的是是方法所在类　指向常量池的索引位置为#4,然后我们发现#4的常量类型是Class，也是符号引用类型，指向常量池#24的位置,而#24是的常量池类型是字面量值为:java/lang/Object

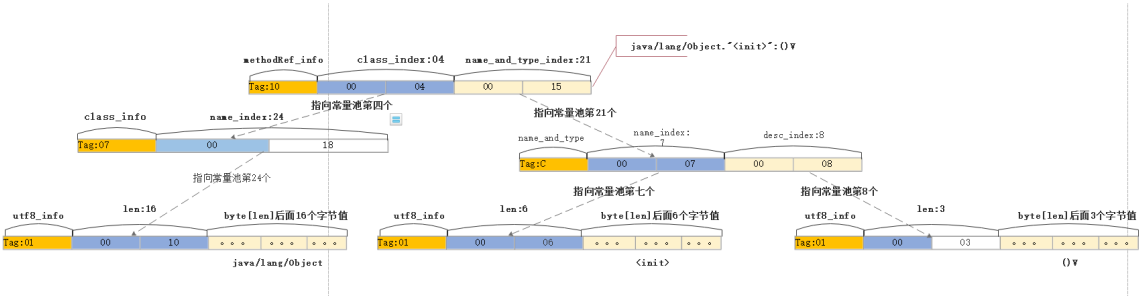
00 15二个字节表示是方法的描述符，指向常量池索引#21的位置,我们发现#21的常量类型是"NameAndType类型"属于引用类型，指向常量池的#7　#8位置

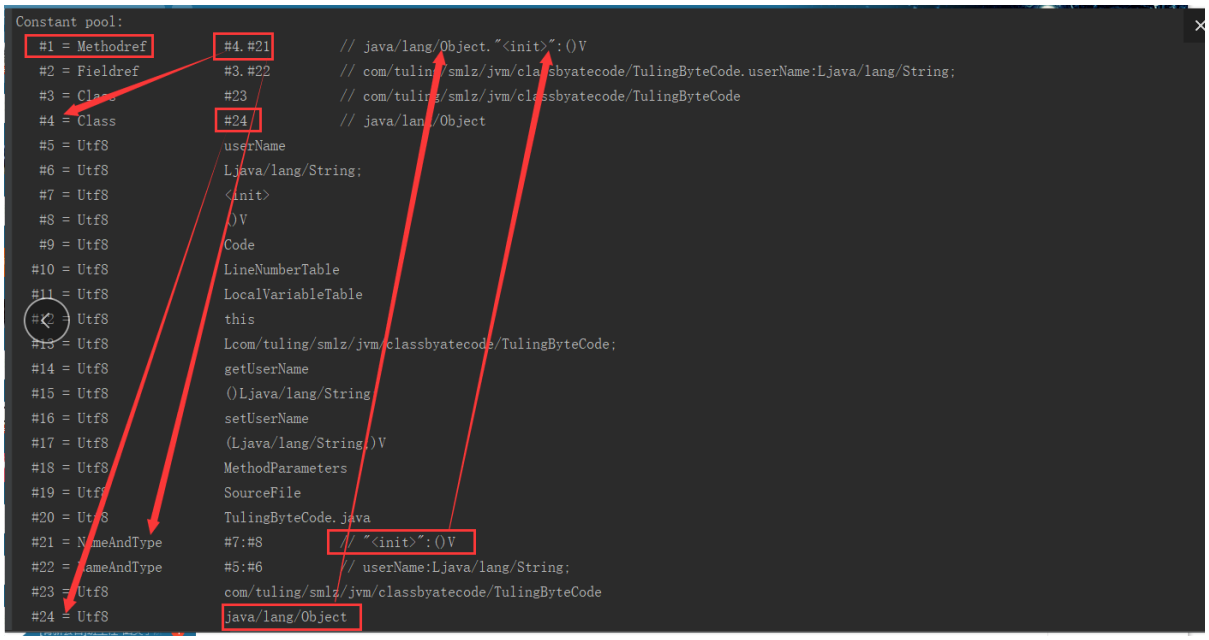
#7常量类型是UTF-8类型属于字面量值为:<init> 为构造方法

#8常量也是UTF-8类型的字面量值为:()V

所以常量池中的第一个常量是：java/lang/Object."<init>":()V

画图分析:





第二个常量分析：09 00 03 00 16

09表示的是我们的 CONSTANT_Methodref_info 字段类型的常量

CONSTANT_Fieldref_info结构

00 03表示的class_index 表示是常量池中第三个 为我们的class常量的索引位置

00 16:表示该字段的名称和类型 指向我们常量池中索引为22的位置

解释:03表示指向常量池第三个位置,我们发现第三个位子是Class类型的常量，03位置的常量池应用指向的是#23的位置，而我们的#23常量池类型是utf-8表示是字面量

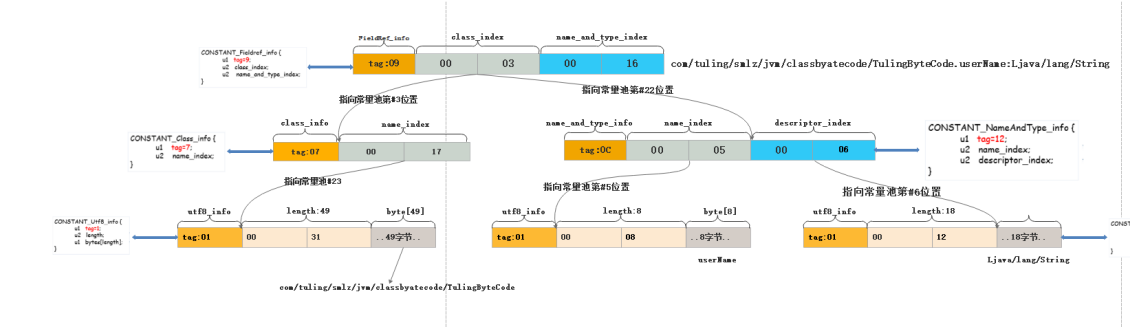
值为:com/tuling/smlz/jvm/classbyatecode/TulingByteCode

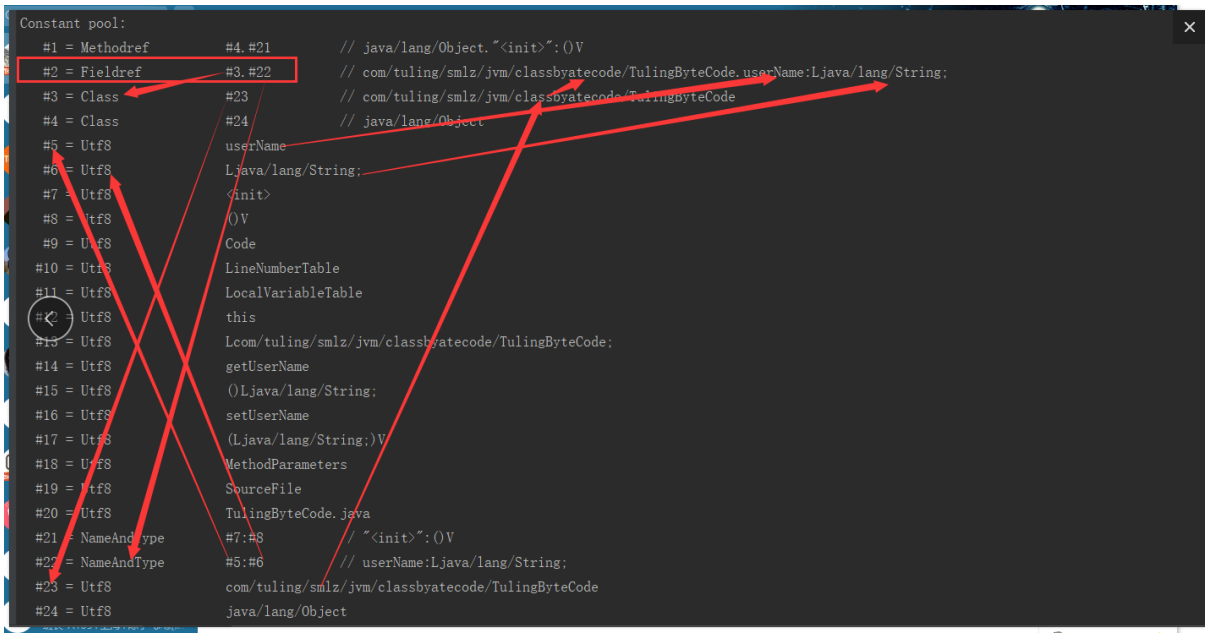
#22为常量池类型的nameAndType类型，分别指向我们的常量池第#5(utf-8类型的常量)的位置表示我们的字段的名称userName，#6指向的是常量池第六个位置,类型是utf-8类型的值为:Ljava/lang/String;

第二个常量

com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;

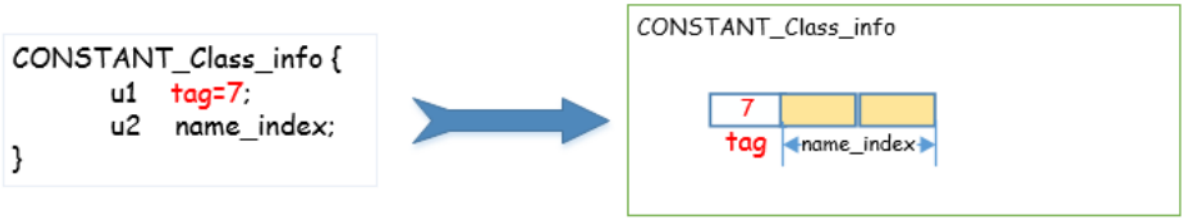
画图分析:





```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第三个常量分析: 07 00 17



第一个字节:07表示的是　class_info符号引用类型的常量

第二三个字节: 00 17表示是指向常量池中索引为23的位置,#23的常量池类型是utf8字面量

那么utf8_info的结构如下:

第#23的常量的结构是

其中 01表示utf8_info的常量类型

00 31：表示后面跟着49个字节是字面量的值

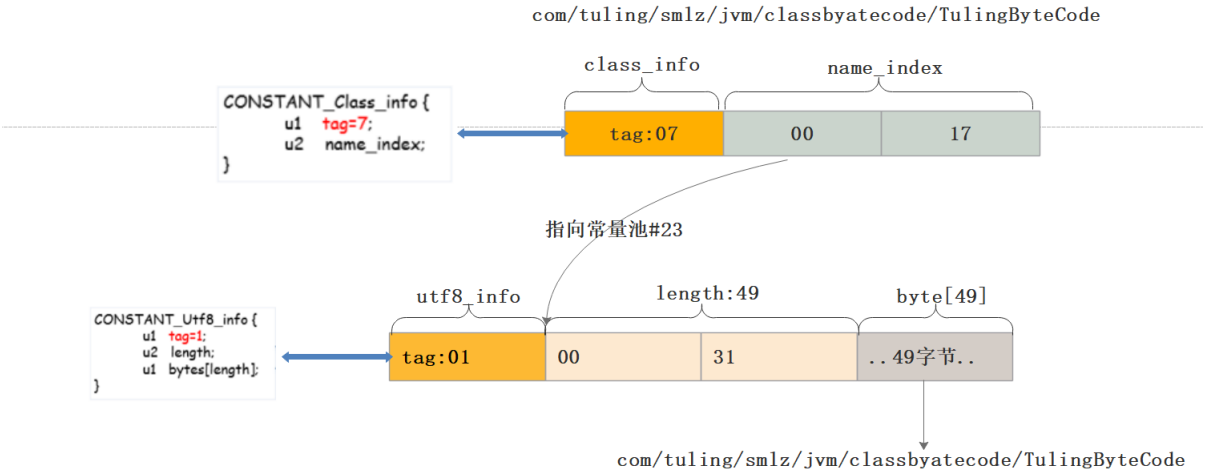
01 00 31 63 6F 6D 2F 74 75 6C 69 6E 67 2F 73 6D 6C

7A 2F 6A 76 6D 2F 63 6C 61 73 73 62 79 61 74 65

63 6F 64 65 2F 54 75 6C 69 6E 67 42 79 74 65 43

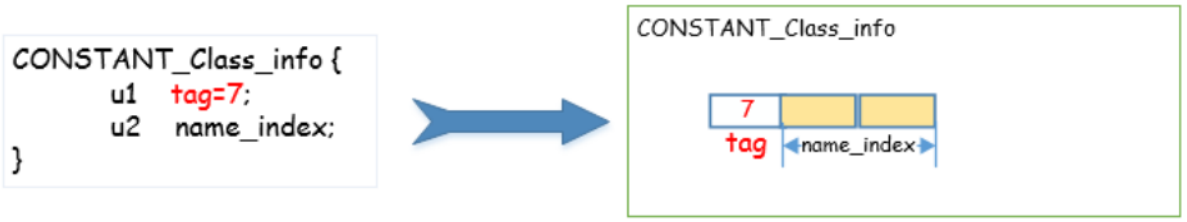
6F 64 65

```
0100  00 10 4D 65 74 68 6F 64   50 61 72 61 6D 65 74 65   ..MethodParamete
0110  72 73 01 00 0A 53 6F 75   72 63 65 46 69 6C 65 01   rs...SourceFile.
0120  00 13 54 75 6C 69 6E 67   42 79 74 65 43 6F 64 65   ..TulingByteCode
0130  2E 6A 61 76 61 0C 00 07   00 08 0C 00 05 00 06 01   .java............
0140  00 31 63 6F 6D 2F 74 75   6C 69 6E 67 2F 73 6D 6C   .1com/tuling/sml
0150  7A 2F 6A 76 6D 2F 63 6C   61 73 73 62 79 61 74 65   z/jvm/classbyate
0160  63 6F 64 65 2F 54 75 6C   69 6E 67 42 79 74 65 43   code/TulingByteC
0170  6F 64 65 01 00 10 6A 61   76 61 2F 6C 61 6E 67 2F   ode...java/lang/
0180  4F 62 6A 65 63 74 00 21   00 03 00 04 00 00 00 01   Object.!........
0190  00 02 00 05 00 06 00 00   00 03 00 01 00 07 00 08   ................
01a0  00 01 00 09 00 00 00 2F   00 01 00 01 00 00 00 05   ......./........
01b0  2A B7 00 01 B1 00 00 00   02 00 0A 00 00 00 06 00   *...............
```




```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第四个常量分析：07 00 18



第一个字节:07表示的是　class_info符号引用类型的常量

第二三个字节: 00 18表示是指向常量池中索引为24的位置,#24的常量池类型是utf8字面量

那么utf8_info的结构如下:

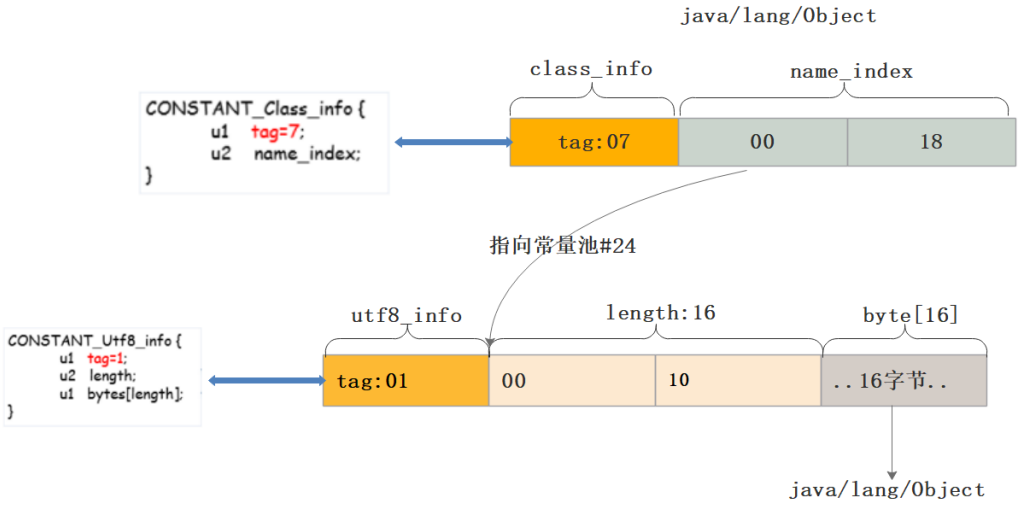
01 00 10 6A 61　76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74

其中 01表示utf8_info的常量类型

00 10：表示后面跟着16个字节是字面量的值

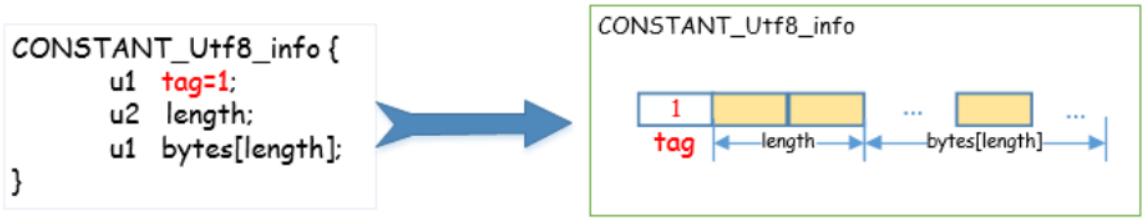
6A 61　76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 字面量的值

为:java/lang/Object



```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第五个常量：01 00 08 75 73 65 72 4E 61 6D 65

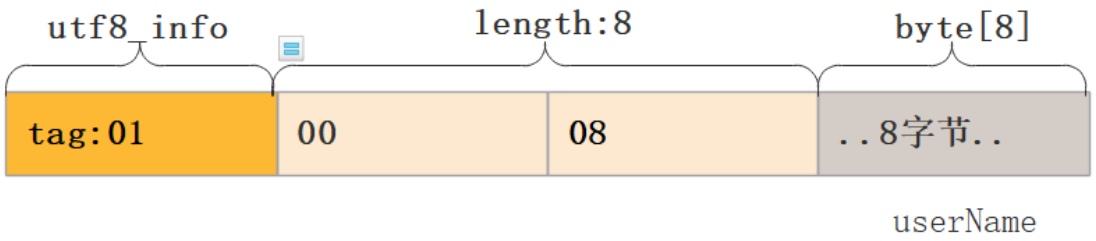


01：tag位表示的是utf8类型的字面量常量

00 08二个字节表示的是字面量常量的长度为8 、

75 73 65 72 4E 61 6D 65 转为字符串为userName

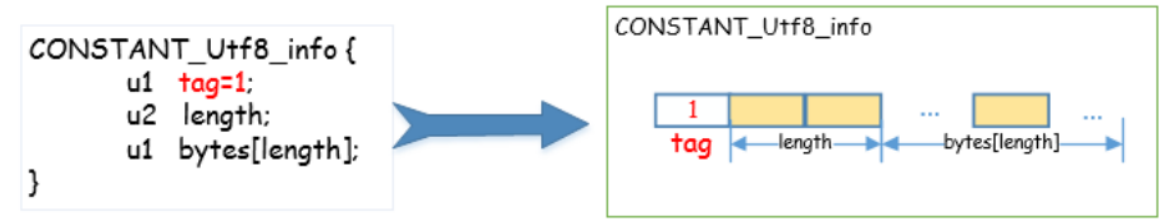
```
CA FE BA BE 00 00 00 34  00 19 0A 00 04 00 15 09   .......4........
00 03 00 16 07 00 17 07  00 18 01 00 08 75 73 65   ..........  ...use
72 4E 61 6D 65 01 00 12  4C 6A 61 76 61 2F 6C 61   rName...Ljava/la
6E 67 2F 53 74 72 69 6E  67 3B 01 00 06 3C 69 6E   ng/String;...<in
```



```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第六个常量分析:

01 00 12 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B



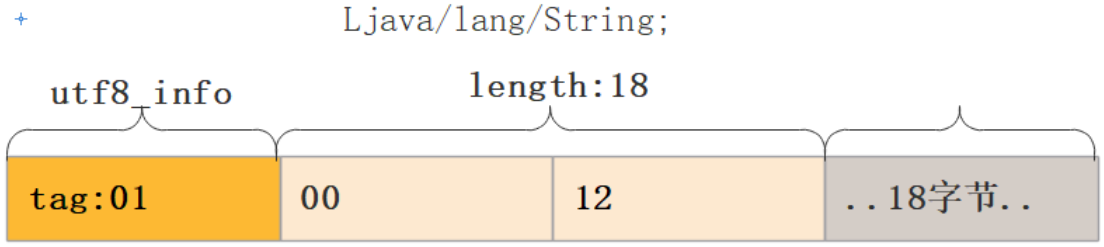
01：tag位表示的是utf8类型的字面量常量

00 12二个字节表示的是字面量常量的长度为18 、

4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 转为字符串为

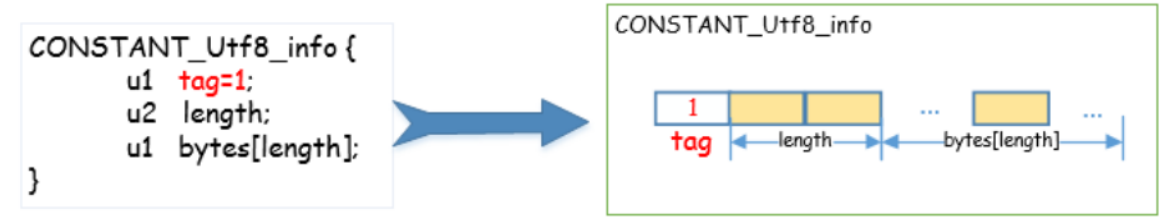
Ljava/lang/String;

```
00  CA FE BA BE 00 00 00 34  00 19 0A 00 04 00 15 09   .......4........
10  00 03 00 16 07 00 17 07  00 18 01 00 08 75 73 65   .............use
20  72 4E 61 6D 65 01 00 12  4C 6A 61 76 61 2F 6C 61   rName...Ljava/la
30  6E 67 2F 53 74 72 69 6E  67 3B 01 00 06 3C 69 6E   ng/String;...<in
40  69 74 3E 01 00 03 28 29  56 01 00 04 43 6F 64 65   it>...()V...Code
```



```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

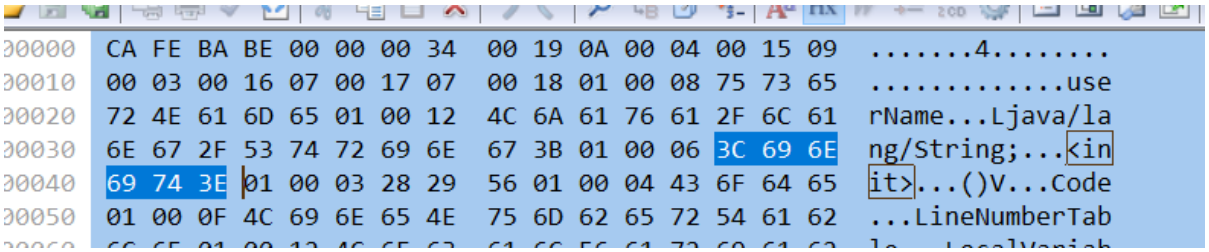
第七个常量分析:01 00 06 3C 69 6E 69 74 3E



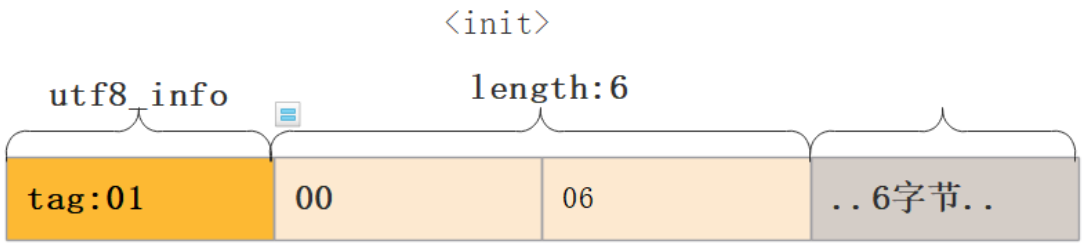
01：tag位表示的是utf8类型的字面量常量

00 06二个字节表示的是字面量常量的长度为6

3C 69 6E 69 74 3E 转为字符串为 <init>

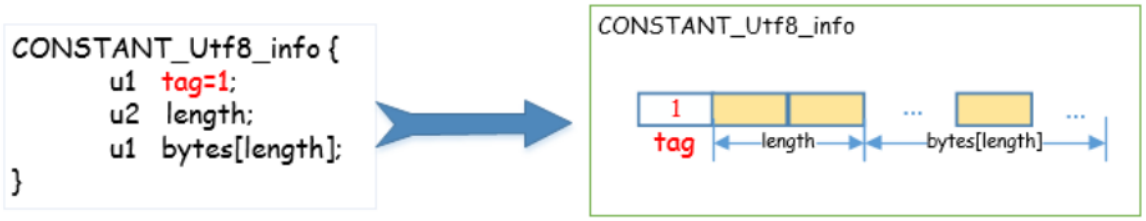


```
00000  CA FE BA BE 00 00 00 34  00 19 0A 00 04 00 15 09   .......4........
00010  00 03 00 16 07 00 17 07  00 18 01 00 08 75 73 65   .............use
00020  72 4E 61 6D 65 01 00 12  4C 6A 61 76 61 2F 6C 61   rName...Ljava/la
00030  6E 67 2F 53 74 72 69 6E  67 3B 01 00 06 3C 69 6E   ng/String;...<in
00040  69 74 3E 01 00 03 28 29  56 01 00 04 43 6F 64 65   it>...()V...Code
00050  01 00 0F 4C 69 6E 65 4E  75 6D 62 65 72 54 61 62   ...LineNumberTab
```



```
Terminal
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第八个常量分析:01 00 03 28 29 56

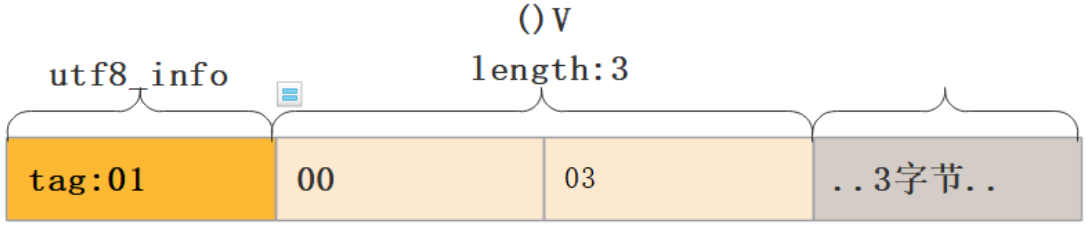


01：tag位表示的是utf8类型的字面量常量

00 03二个字节表示的是字面量常量的长度为3

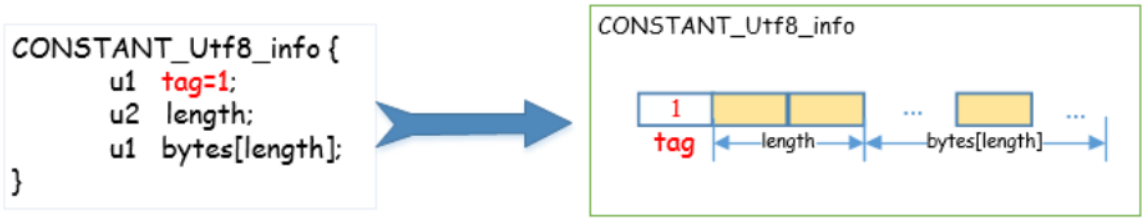
28 29 56 转为字符串为 ()V

```
00 03 00 16 07 00 17 07  00 18 01 00 08 75 73 65  ............use
72 4E 61 6D 65 01 00 12  4C 6A 61 76 61 2F 6C 61  rName...Ljava/la
6E 67 2F 53 74 72 69 6E  67 3B 01 00 06 3C 69 6E  ng/String;...<in
69 74 3E 01 00 03 28 29  56 01 00 04 43 6F 64 65  it>...()V...Code
01 00 0F 4C 69 6E 65 4E  75 6D 62 65 72 54 61 62  ...LineNumberTab
6C 65 01 00 12 4C 6F 63  61 6C 56 61 72 69 61 62  le...LocalVariab
6C 65 54 61 62 6C 65 01  00 04 74 68 69 73 01 00  leTable...this..
```



```
Terminal
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userNa
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第九个常量分析:01 00 04 43 6F 64 65

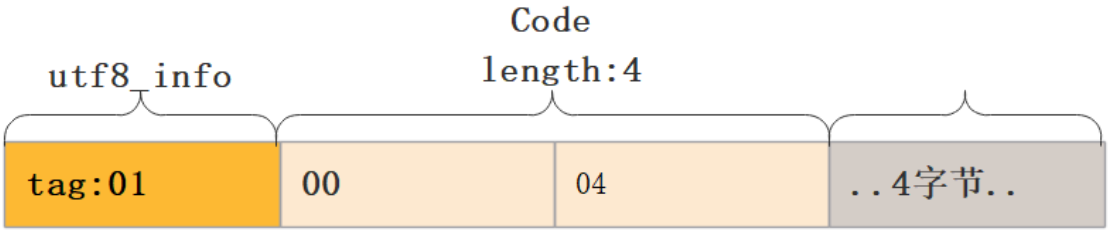


01：tag位表示的是utf8类型的字面量常量

00 04二个字节表示的是字面量常量的长度为4

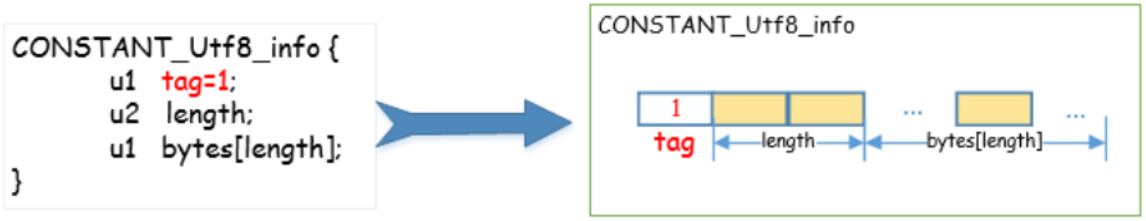
43 6F 64 65 转为字符串为 Code

```
CA FE BA BE 00 00 00 34  00 19 0A 00 04 00 15 09  .......4........
00 03 00 16 07 00 17 07  00 18 01 00 08 75 73 65  .............use
72 4E 61 6D 65 01 00 12  4C 6A 61 76 61 2F 6C 61  rName...Ljava/la
6E 67 2F 53 74 72 69 6E  67 3B 01 00 06 3C 69 6E  ng/String;...<in
69 74 3E 01 00 03 28 29  56 01 00 04 43 6F 64 65  it>...()V...Code
01 00 0F 4C 69 6E 65 4E  75 6D 62 65 72 54 61 62  ...LineNumberTab
6C 65 01 00 12 4C 6F 63  61 6C 56 61 72 69 61 62  le...LocalVariab
6C 65 54 61 62 6C 65 01  00 04 74 68 69 73 01 00  leTable...this..
```




```
Terminal
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第十个常量分析:01 00 0F 4C 69 6E 65 4E 75 6D 62 65 72 54 61 62 6C 65



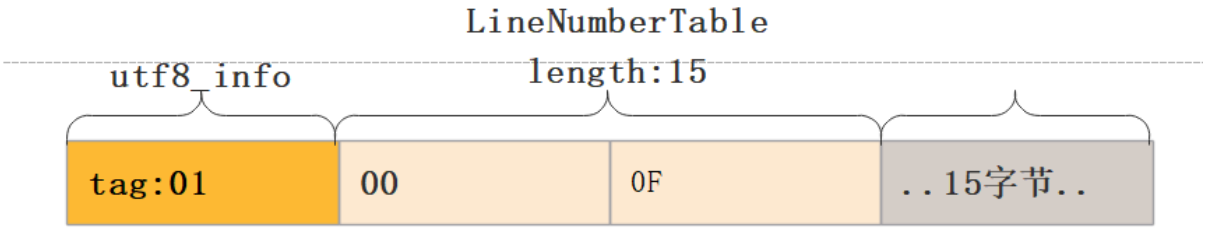
01：tag位表示的是utf8类型的字面量常量

00 0F二个字节表示的是字面量常量的长度为15

4C 69 6E 65 4E 75 6D 62 65 72 54 61 62 6C 65 转为字符串为 LineNumberTable

表示这个是行号表

```
69 74 3E 01 00 03 28 29  56 01 00 04 43 6F 64 65  it>...()V...Code
01 00 0F 4C 69 6E 65 4E  75 6D 62 65 72 54 61 62  ...LineNumberTab
6C 65 01 00 12 4C 6F 63  61 6C 56 61 72 69 61 62  le...LocalVariab
6C 65 54 61 62 6C 65 01  00 04 74 68 69 73 01 00  leTable...this..
33 4C 63 6F 6D 2F 74 75  6C 69 6E 67 2F 73 6D 6C  3Lcom/tuling/sml
```




```
Terminal
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第11个常量:01 00 12 4C 6F 63 61 6C 56 61 72 69 61 62 6C 65 54 61 62 6C 65

01：tag位表示的是utf8类型的字面量常量

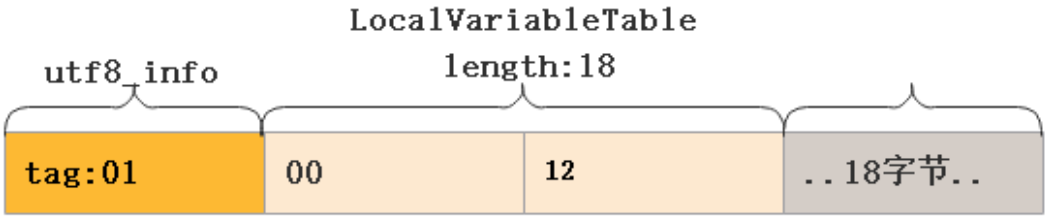
00 12二个字节表示的是字面量常量的长度为18

4C 6F 63 61 6C 56 61 72 69 61 62 6C 65 54 61 62 6C 65 转为字符串为

LocalVariableTable

表示这个是本地变量表

```
50  01 00 0F 4C 69 6E 65 4E  75 6D 62 65 72 54 61 62  ...LineNumberTab
60  6C 65 01 00 12 4C 6F 63  61 6C 56 61 72 69 61 62  le...LocalVariab
70  6C 65 54 61 62 6C 65 01  00 04 74 68 69 73 01 00  leTable...this..
80  33 4C 63 6F 6D 2F 74 75  6C 69 6E 67 2F 73 6D 6C  3Lcom/tuling/sml
```



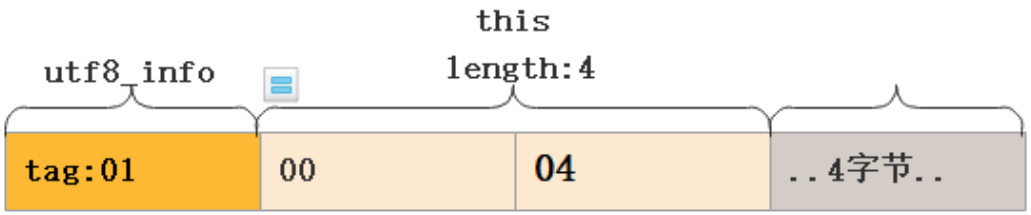

```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第12个常量: 01 00 04 74 68 69 73

01：tag位表示的是utf8类型的字面量常量

00 14二个字节表示的是字面量常量的长度为4

74 68 69 73　转为字符串为this



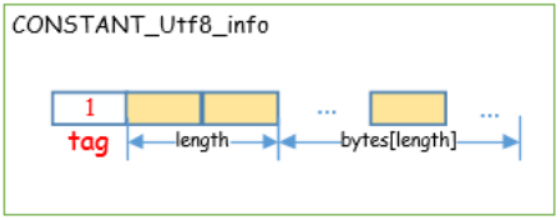

```
01 00 0F 4C 69 6E 65 4E  75 6D 62 65 72 54 61 62  ...LineNumberTab
6C 65 01 00 12 4C 6F 63  61 6C 56 61 72 69 61 62  le...LocalVariab
6C 65 54 61 62 6C 65 01  00 04 74 68 69 73 01 00  leTable...this..
33 4C 63 6F 6D 2F 74 75  6C 69 6E 67 2F 73 6D 6C  3Lcom/tuling/sml
7A 2F 6A 76 6D 2F 63 6C  61 73 73 62 79 61 74 65  z/jvm/classbyate
63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/TulingByteC
6F 64 65 3B 01 00 0B 67  65 74 55 73 65 72 4E 61  ode;...getUserNa
```

```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第13个常量:

01 00 33 4C 63 6F 6D 2F 74 75 6C 69 6E 67 2F 73 6D 6C 7A 2F 6A 76 6D 2F 63 6C 61 73 73 62 79 61 74 65 63 6F 64 65 2F 54 75 6C 69 6E 67 42 79 74 65 43 6F 64 65 3B

```
CONSTANT_Utf8_info {
        u1  tag=1;
        u2  length;
        u1  bytes[length];
}
```



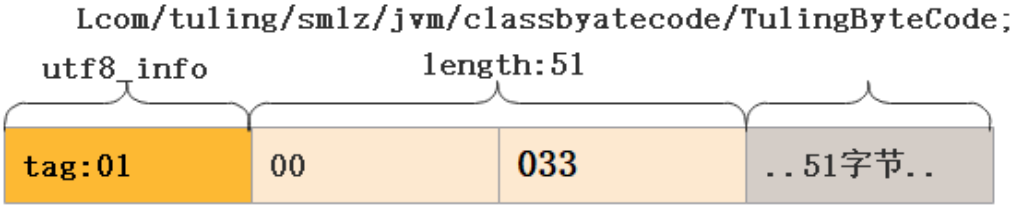
01：tag位表示的是utf8类型的字面量常量

00 33二个字节表示的是字面量常量的长度为51

4C 63 6F 6D 2F 74 75 6C 69 6E 67 2F 73 6D 6C 7A 2F 6A 76 6D 2F 63 6C 61 73 73 62 79 61 74 65 63 6F 64 65 2F 54 75 6C 69 6E 67 42 79 74 65 43 6F 64 65 3B

表示字符串: Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;

```
60  6C 65 01 00 12 4C 6F 63  61 6C 56 61 72 69 61 62  le...LocalVariab
70  6C 65 54 61 62 6C 65 01  00 04 74 68 69 73 01 00  leTable...this..
80  33 4C 63 6F 6D 2F 74 75  6C 69 6E 67 2F 73 6D 6C  3Lcom/tuling/sml
90  7A 2F 6A 76 6D 2F 63 6C  61 73 73 62 79 61 74 65  z/jvm/classbyate
a0  63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/TulingByteC
b0  6F 64 65 3B 01 00 0B 67  65 74 55 73 65 72 4E 61  ode;...getUserNa
c0  6D 65 01 00 14 28 29 4C  6A 61 76 61 2F 6C 61 6E  me...()Ljava/lan
d0  67 2F 53 74 72 69 6E 67  3B 01 00 0B 73 65 74 55  g/String;...setU
```




```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

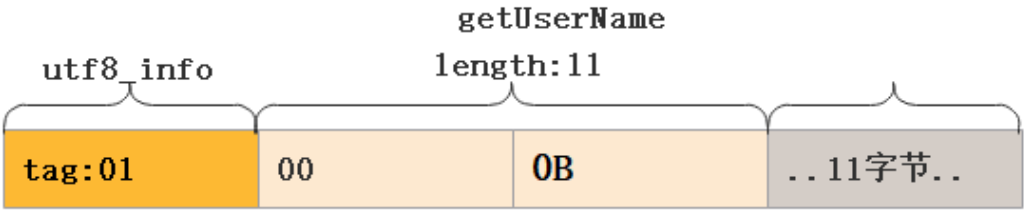
第14个常量:01 00 0B 67 65 74 55 73 65 72 4E 61 6D 65

01：tag位表示的是utf8类型的字面量常量

00 0B 二个字节表示的是字面量常量的长度为11

67 65 74 55 73 65 72 4E 61 6D 65 表示的是字符串getUserName

```
00a0  63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/TulingByteC
00b0  6F 64 65 3B 01 00 0B 67  65 74 55 73 65 72 4E 61  ode;...getUserNa
00c0  6D 65 01 00 14 28 29 4C  6A 61 76 61 2F 6C 61 6E  me...()Ljava/lan
00d0  67 2F 53 74 72 69 6E 67  3B 01 00 0B 73 65 74 55  g/String;...setU
00e0  73 65 72 4E 61 6D 65 01  00 15 28 4C 6A 61 76 61  serName...(Ljava
```




```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第15个常量分析:01 00 14 28 29 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B

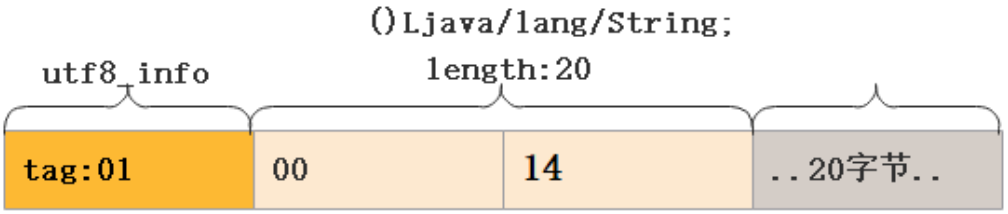
01：tag位表示的是utf8类型的字面量常量

00 14 二个字节表示的是字面量常量的长度为20

接下来20个字节: 28 29 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 表示字符串

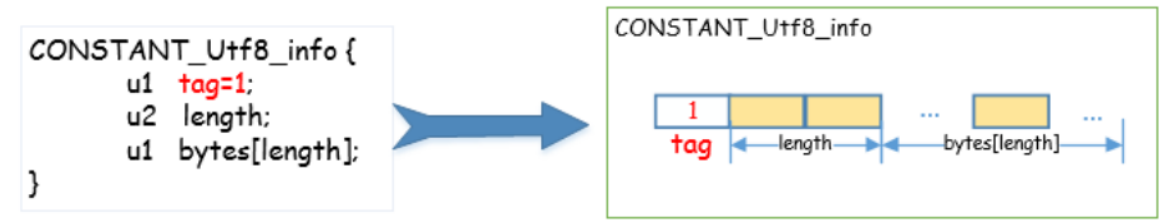
()Ljava/lang/String;

```
b0  6F 64 65 3B 01 00 0B 67  65 74 55 73 65 72 4E 61  ode;...getUserNa
c0  6D 65 01 00 14 28 29 4C  6A 61 76 61 2F 6C 61 6E  me...()Ljava/lan
d0  67 2F 53 74 72 69 6E 67  3B 01 00 0B 73 65 74 55  g/String;...setU
e0  73 65 72 4E 61 6D 65 01  00 15 28 4C 6A 61 76 61  serName...(Ljava
f0  2F 6C 61 6E 67 2F 53 74  72 69 6E 67 3B 29 56 01  /lang/String;)V.
```




```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/S
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第16个常量池分析: 01 00 0B 73 65 74 55 73 65 72 4E 61 6D 65

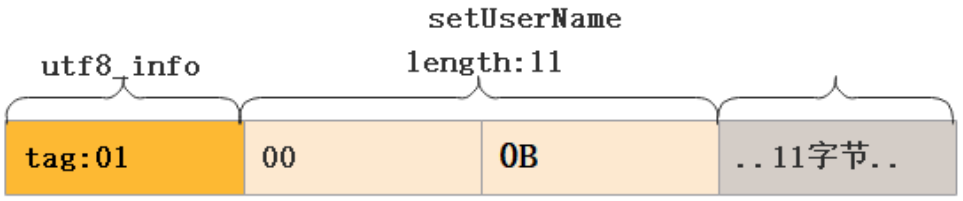


01：tag位表示的是utf8类型的字面量常量

00 0B 二个字节表示的是字面量常量的长度为11

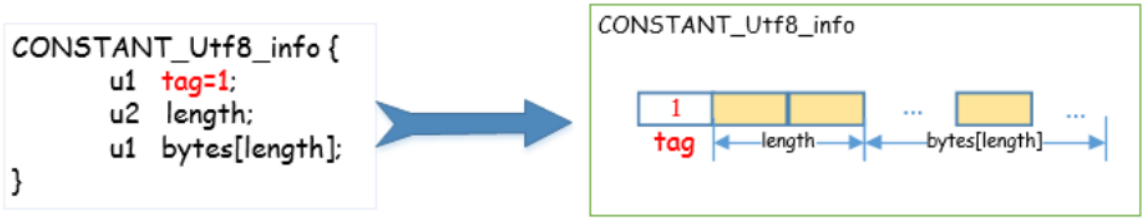
接下来11个字节: 73 65 74 55 73 65 72 4E 61 6D 65 表示字符串 setUserName

```
c0  6D 65 01 00 14 28 29 4C  6A 61 76 61 2F 6C 61 6E  me...()Ljava/lan
d0  67 2F 53 74 72 69 6E 67  3B 01 00 0B 73 65 74 55  g/String;...setU
e0  73 65 72 4E 61 6D 65 01  00 15 28 4C 6A 61 76 61  serName...(Ljava
f0  2F 6C 61 6E 67 2F 53 74  72 69 6E 67 3B 29 56 01  /lang/String;)V.
```




```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第17个常量池:01 00 15 28 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 29 56



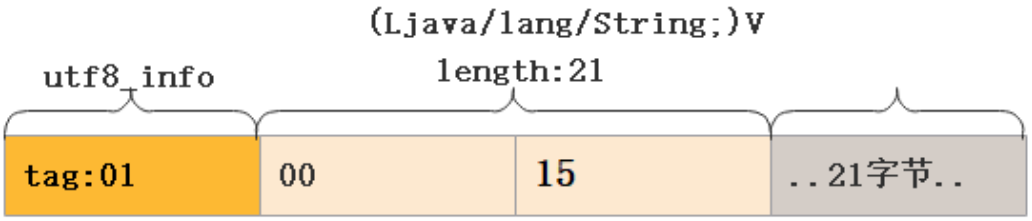
01：tag位表示的是utf8类型的字面量常量

00 15 二个字节表示的是字面量常量的长度为21

接下来21个字节: 28 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 29 56

表示字符串 (Ljava/lang/String;)V

```
d0  67 2F 53 74 72 69 6E 67  3B 01 00 0B 73 65 74 55  g/String;...setU
e0  73 65 72 4E 61 6D 65 01  00 15 28 4C 6A 61 76 61  serName...(Ljava
f0  2F 6C 61 6E 67 2F 53 74  72 69 6E 67 3B 29 56 01  /lang/String;)V.
00  00 10 4D 65 74 68 6F 64  50 61 72 61 6D 65 74 65  ..MethodParamete
```




```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

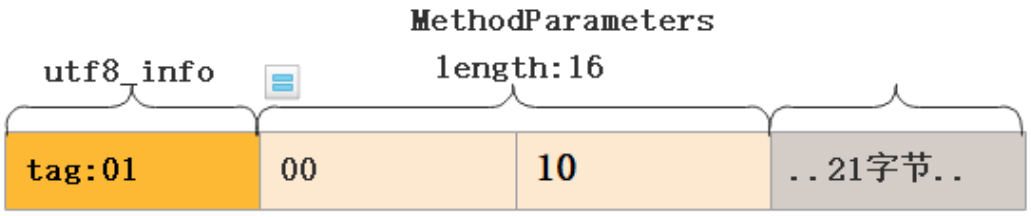
第18个常量：01 00 10 4D 65 74 68 6F 64 50 61 72 61 6D 65 74 65 72 73

01：tag位表示的是utf8类型的字面量常量

00 10 二个字节表示的是字面量常量的长度为16

接下来16个字节：4D 65 74 68 6F 64 50 61 72 61 6D 65 74 65 72 73 表示字符串MethodParameters

```
2F 6C 61 6E 67 2F 53 74  72 69 6E 67 3B 29 56 01  /lang/String;)V.
00 10 4D 65 74 68 6F 64  50 61 72 61 6D 65 74 65  ..MethodParamete
72 73 01 00 0A 53 6F 75  72 63 65 46 69 6C 65 01  rs...SourceFile.
```




```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/S
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

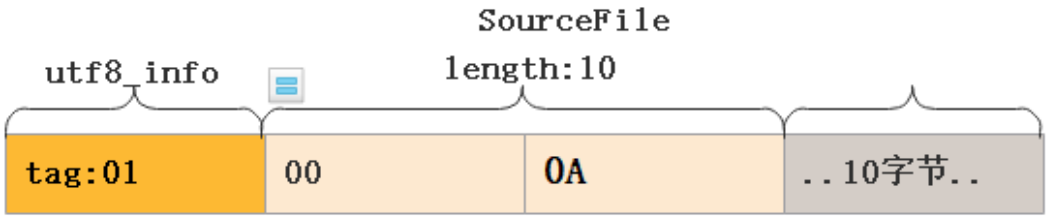
第19个常量: 01 00 0A 53 6F 75 72 63 65 46 69 6C 65

01：tag位表示的是utf8类型的字面量常量

00 0A 二个字节表示的是字面量常量的长度为10

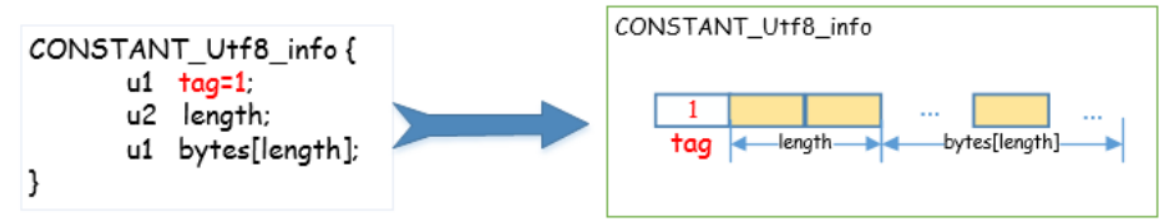
接下来10个字节: 53 6F 75 72 63 65 46 69 6C 65 表示字符串SourceFile

```
00 10 4D 65 74 68 6F 64  50 61 72 61 6D 65 74 65  ..MethodParamete
72 73 01 00 0A 53 6F 75  72 63 65 46 69 6C 65 01  rs...SourceFile.
00 13 54 75 6C 69 6E 67  42 79 74 65 43 6F 64 65  ..TulingByteCode
2E 6A 61 76 61 0C 00 07  00 08 0C 00 05 00 06 01  .java..........
```



```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userNa
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第20个常量分析:01 00 13 54 75 6C 69 6E 67 42 79 74 65 43 6F 64 65 2E 6A 61 76 61

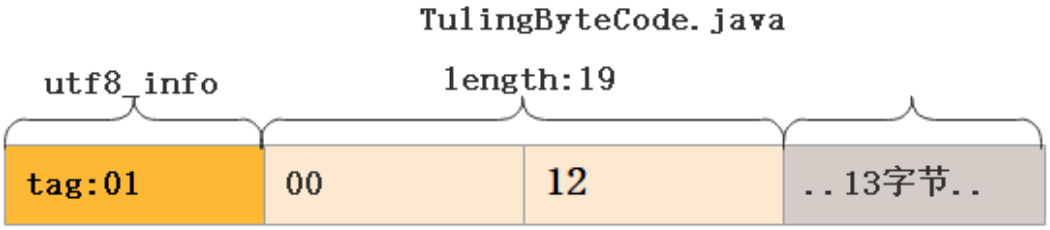


01：tag位表示的是utf8类型的字面量常量

00 13 二个字节表示的是字面量常量的长度为19

接下来19个字节: 54 75 6C 69 6E 67 42 79 74 65 43 6F 64 65 2E 6A 61 76 61 表示字符串TulingByteCode.java

```
0  00 10 4D 65 74 68 6F 64  50 61 72 61 6D 65 74 65  ..MethodParamete
0  72 73 01 00 0A 53 6F 75  72 63 65 46 69 6C 65 01  rs...SourceFile.
0  00 13 54 75 6C 69 6E 67  42 79 74 65 43 6F 64 65  ..TulingByteCode
0  2E 6A 61 76 61 0C 00 07  00 08 0C 00 05 00 06 01  .java..........
0  00 31 63 6F 6D 2F 74 75  6C 69 6E 67 2F 73 6D 6C  .1com/tuling/sml
```




```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

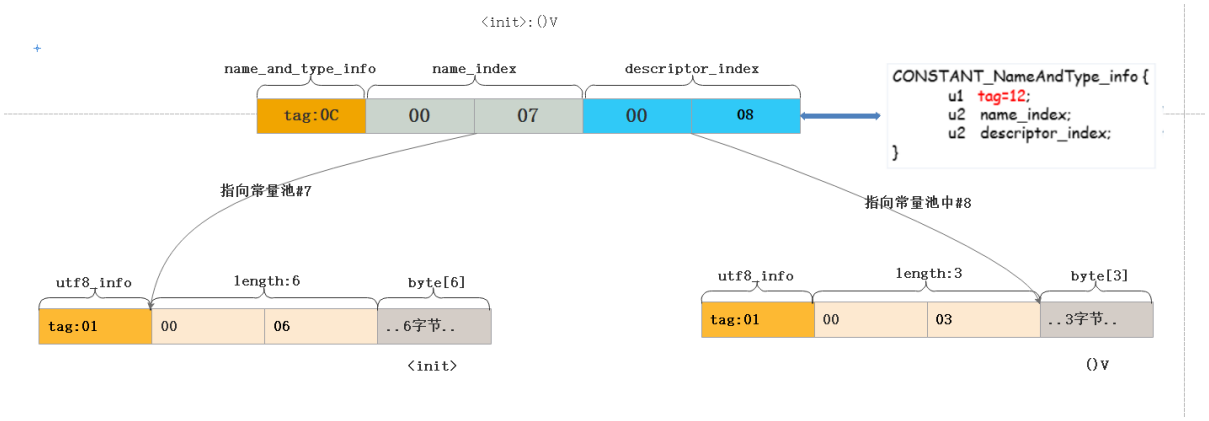
21个常量池分析: 0C 00 07 00 08

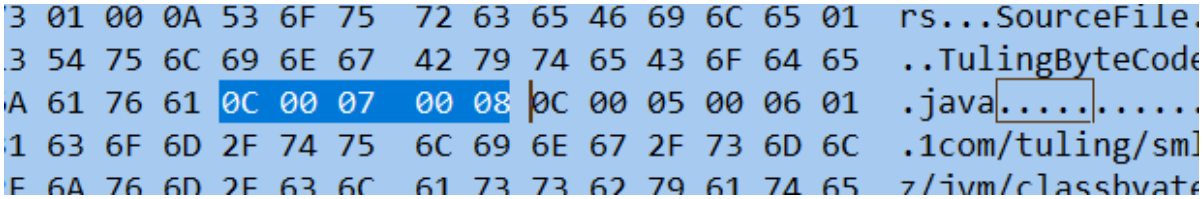
```
CONSTANT_NameAndType_info {
        u1  tag=12;
        u2  name_index;
        u2  descriptor_index;
}
```

0C：tag位表示的是符号引用 nameAndType_info类型的

00 07 指向索引为7的常量池#7

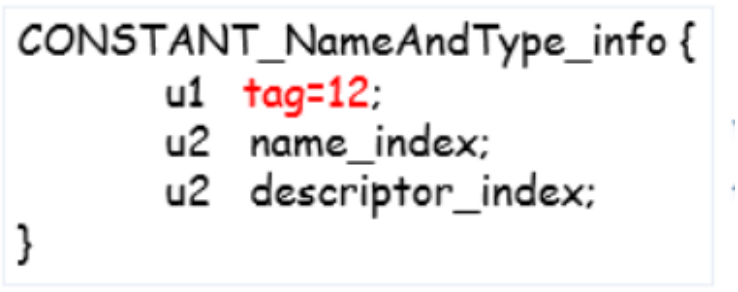
00 08 指向常量池8的位置#8






```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

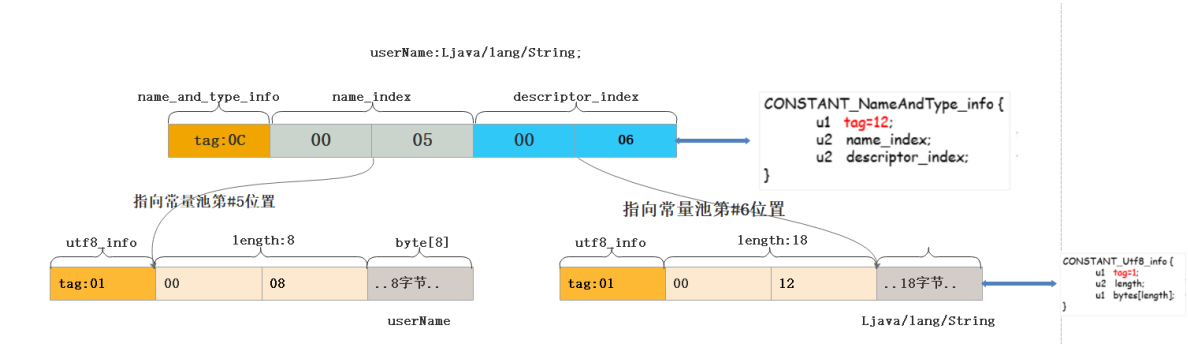
第22个常量池:0C 00 05 00 06



0C：tag位表示的是符号引用 nameAndType_info类型的

00 05 指向索引为7的常量池#5

00 06 指向常量池8的位置#6

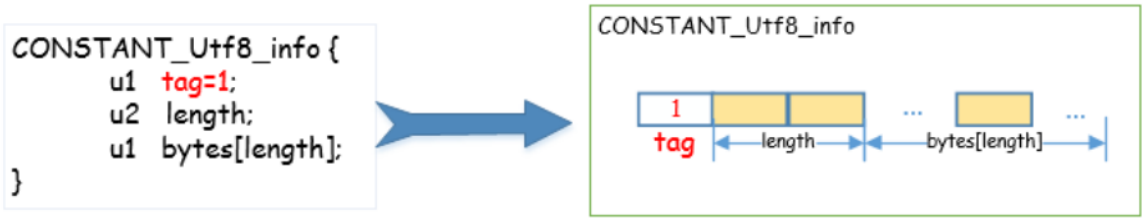



```
00 13 54 75 6C 69 6E 67   42 79 74 65 43 6F 64 65   ..TulingByteCode
2E 6A 61 76 61 0C 00 07   00 08 0C 00 05 00 06 01   .java.....[.....].
00 31 63 6F 6D 2F 74 75   6C 69 6E 67 2F 73 6D 6C   .1com/tuling/sml
7A 2F 6A 76 6D 2F 63 6C   61 73 73 62 79 61 74 65   z/jvm/classbyate
```

```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第23个常量池:

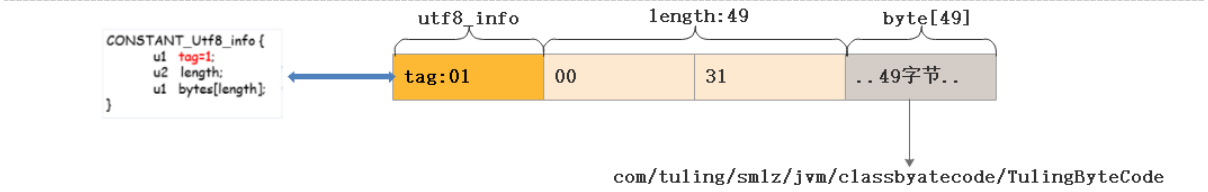
01 00 31 63 6F 6D 2F 74 75 6C 69 6E 67 2F 73 6D 6C 7A 2F 6A 76 6D 2F 63 6C 61 73 73 62 79 61 74 65 63 6F 64 65 2F 54 75 6C 69 6E 67 42 79 74 65 43 6F 64 65



01：tag位表示的是utf8类型的字面量常量

00 31 二个字节表示的是字面量常量的长度为49

接下来49个字节: 63 6F 6D 2F 74 75 6C 69 6E 67 2F 73 6D 6C 7A 2F 6A 76 6D 2F 63 6C 61 73 73 62 79 61 74 65 63 6F 64 65 2F 54 75 6C 69 6E 67 42 79 74 65 43 6F 64 65 表示字符串com/tuling/smlz/jvm/classbyatecode/TulingByteCode




```
00 13 54 75 6C 69 6E 67  42 79 74 65 43 6F 64 65  ..TulingByteCode
2E 6A 61 76 61 0C 00 07  00 08 0C 00 05 00 06 01  .java..........
00 31 63 6F 6D 2F 74 75  6C 69 6E 67 2F 73 6D 6C  .1com/tuling/sml
7A 2F 6A 76 6D 2F 63 6C  61 73 73 62 79 61 74 65  z/jvm/classbyate
63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/TulingByteC
6F 64 65 01 00 10 6A 61  76 61 2F 6C 61 6E 67 2F  ode...java/lang/
4F 62 6A 65 63 74 00 21  00 03 00 04 00 00 00 01  Object.!........
00 02 00 05 00 06 00 00  00 03 00 01 00 07 00 08  ................
```

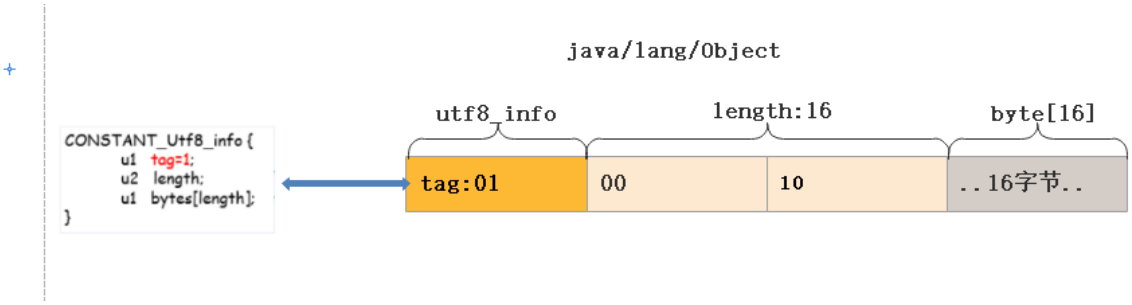
```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

第24个常量池:01 00 10 6A 61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74

01：tag位表示的是utf8类型的字面量常量

00 10 二个字节表示的是字面量常量的长度为16

接下来16个字节：6A 61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 表示字符串java/lang/Object




```
50  7A 2F 6A 76 6D 2F 63 6C  61 73 73 62 79 61 74 65  z/jvm/classbyate
60  63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/TulingByteC
70  6F 64 65 01 00 10 6A 61  76 61 2F 6C 61 6E 67 2F  ode...java/lang/
80  4F 62 6A 65 63 74 00 21  00 03 00 04 00 00 00 01  Object.!........
90  00 02 00 05 00 06 00 00  00 03 00 01 00 07 00 08  ................
a0  00 01 00 09 00 00 00 2F  00 01 00 01 00 00 00 05  ......./........
```

```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
   #5 = Utf8               userName
   #6 = Utf8               Ljava/lang/String;
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
  #15 = Utf8               ()Ljava/lang/String;
  #16 = Utf8               setUserName
  #17 = Utf8               (Ljava/lang/String;)V
  #18 = Utf8               MethodParameters
  #19 = Utf8               SourceFile
  #20 = Utf8               TulingByteCode.java
  #21 = NameAndType        #7:#8          // "<init>":()V
  #22 = NameAndType        #5:#6          // userName:Ljava/lang/String;
  #23 = Utf8               com/tuling/smlz/jvm/classbyatecode/TulingByteCode
  #24 = Utf8               java/lang/Object
```

四:Class文件结构访问标识符号解析 Access_flag

解析我们的class文件是类还是接口，是否定义为public的,是否是abstract,是否被final修饰。

类的访问权限查询手册		
flag_name	value	desc
ACC_PUBLIC	0x0001	public修饰符号
ACC_FINAL	0x0010	没有子类
ACC_SUPER	0x0020	通过invokeSpecial指令可以调用父类的方法
ACC_INTERFACE	0x0200	标识是一个接口
ACC_ABSTRACT	0x0400	表示是一个抽象类
ACC_SYNTHETIC	0x1000	该class是动态生成的没有源文件
ACC_ANNOTATION	0x2000	是一个注解类型
ACC_ENUM	0x4000	表示是一个枚举类型
ACC_PRIVATE	0x0002	表示私有的

访问标志符号占用二个字节: 00 21

```
00150  7A 2F 6A 76 6D 2F 63 6C  61 73 73 62 79 61 74 65  z/jvm/classbyate
00160  63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/TulingByteC
00170  6F 64 65 01 00 10 6A 61  76 61 2F 6C 61 6E 67 2F  ode...java/lang/
00180  4F 62 6A 65 63 74 00 21  00 03 00 04 00 00 00 01  Object.!........
00190  00 02 00 05 00 06 00 00  00 03 00 01 00 07 00 08  ................
001a0  00 01 00 09 00 00 00 2F  00 01 00 01 00 00 00 05  ......./........
```

我们发现这个class文件的访问标识字节是0x0021，我们去查询手册中查询没有这个对应的?

原因:jvm规范并没有穷举出所以的类型 而是通过位运算的出来的.

0x0021 = 0x0020 位运算 0x0001 那么我们可以得出这个class的访问权限是ACC_PUBLIC和ACC_SUPER

```
Classfile /D:/work_space/idea_space/spring-cloud-source/tuling-jvm/target/classes/com/tuling
  Last modified 2019-11-7; size 629 bytes
  MD5 checksum 9034537784194e8b1e1f7efc00f0aecb
  Compiled from "TulingByteCode.java"
public class com.tuling.smlz.jvm.classbyatecode.TulingByteCode
  minor version: 0
  major version: 52
  flags: ACC_PUBLIC, ACC_SUPER
Constant pool:
```

五: This class name的描述当前的所属类

this class name 占用二个字节:00 03 表示索引 指向的是常量池中的第三个常量

```
50  7A 2F 6A 76 6D 2F 63 6C  61 73 73 62 79 61 74 65  z/jvm/classbyate
60  63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/TulingByteC
70  6F 64 65 01 00 10 6A 61  76 61 2F 6C 61 6E 67 2F  ode...java/lang/
80  4F 62 6A 65 63 74 00 21  00 03 00 04 00 00 00 01  Object.!........
90  00 02 00 05 00 06 00 00  00 03 00 01 00 07 00 08  ................
a0  00 01 00 09 00 00 00 2F  00 01 00 01 00 00 00 05  ......./........
b0  2A B7 00 01 B1 00 00 00  02 00 0A 00 00 00 06 00  *...............
```

根据第三部分常量池分析得出第三个常量分析得出如下

```
#2 = Fieldref           #3.#22          // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
#3 = Class              #23             // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
#4 = Class              #24             // java/lang/Object
#5 = Utf8               userName
#6 = Utf8               Ljava/lang/String;
#7 = Utf8               <init>
#8 = Utf8               ()V
#9 = Utf8               Code
#10 = Utf8              LineNumberTable
#11 = Utf8              LocalVariableTable
#12 = Utf8              this
#13 = Utf8              Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
#14 = Utf8              getUserName
#15 = Utf8              ()Ljava/lang/String;
#16 = Utf8              setUserName
#17 = Utf8              (Ljava/lang/String;)V
#18 = Utf8              MethodParameters
#19 = Utf8              SourceFile
#20 = Utf8              TulingByteCode.java
#21 = NameAndType       #7:#8           // "<init>":()V
#22 = NameAndType       #5:#6           // userName:Ljava/lang/String;
#23 = Utf8              com/tuling/smlz/jvm/classbyatecode/TulingByteCode
```

所以我们的this class name：表示当前类

com/tuling/smlz/jvm/classbyatecode/TulingByteCode

第六部分: super class name （当前class的父类名字）

同样占用二个字节:00 04 也是表示索引值，指向常量池中第四个常量

```
50  7A 2F 6A 76 6D 2F 63 6C  61 73 73 62 79 61 74 65  z/jvm/class
60  63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/Tuling
70  6F 64 65 01 00 10 6A 61  76 61 2F 6C 61 6E 67 2F  ode...java/
80  4F 62 6A 65 63 74 00 21  00 03 00 04 00 00 00 01  Object.!..
90  00 02 00 05 00 06 00 00  00 03 00 01 00 07 00 08  ..........
a0  00 01 00 09 00 00 00 2F  00 01 00 01 00 00 00 05  ......./...
```

根据第三部分常量池的分析第四个常量池得出.

```
#4 = Class              #24            // java/lang/Object
#5 = Utf8               userName
#6 = Utf8               Ljava/lang/String;
#7 = Utf8               <init>
#8 = Utf8               ()V
#9 = Utf8               Code
#10 = Utf8              LineNumberTable
#11 = Utf8              LocalVariableTable
#12 = Utf8              this
#13 = Utf8              Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
#14 = Utf8              getUserName
#15 = Utf8              ()Ljava/lang/String;
#16 = Utf8              setUserName
#17 = Utf8              (Ljava/lang/String;)V
#18 = Utf8              MethodParameters
#19 = Utf8              SourceFile
#20 = Utf8              TulingByteCode.java
#21 = NameAndType       #7:#8          // "<init>":()V
#22 = NameAndType       #5:#6          // userName:Ljava/lang/String;
#23 = Utf8              com/tuling/smlz/jvm/classbyatecode/TulingByteCode
#24 = Utf8              java/lang/Object
```

所以我们的super class name表示的意思是: java/lang/Object

第七部分:接口信息（**标注 我们的当前class没有实现接口为了演示效果我用的另外一个类演示） 这个类我们实现了二个接口 分别为ITulingIntf ITulingIntf

```
/**
 * Created by smlz on 2019/11/7.
 */
public class TulingCompent implements ITulingIntf,ITulingIntf2 {
```

00 02 00 08 00 09 这六个字节描述的信息是

```
0140  74 01 00 15 4C 6A 61 76  61 2F 69 6F 2F 50 72 69  t...Lj
0150  6E 74 53 74 72 65 61 6D  3B 01 00 13 6A 61 76 61  ntStre
0160  2F 69 6F 2F 50 72 69 6E  74 53 74 72 65 61 6D 01  /io/Pr
0170  00 07 70 72 69 6E 74 6C  6E 00 21 00 06 00 07 00  ..prir
0180  02 00 08 00 09 00 01 00  02 00 0A 00 0B 00 00 00  .....
0190  05 00 01 00 0C 00 0D 00  01 00 0E 00 00 00 2F 00  ......
```

00 02表示我们实现了几个接口 这里很明星我们是实现了二个接口

00 08(第一个接口) 表示的是接口的位于常量池中的索引.#8

```
#8 = Class              #37            // com/tuling/smlz/jvm/classbyatecode/ITulingIntf
#37 = Utf8              com/tuling/smlz/jvm/classbyatecode/ITulingIntf
```

所以00 08指向的接口是:com/tuling/smlz/jvm/classbyatecode/ITulingIntf

00 09（第二个接口）表示的是接口的位于常量池中的索引#9

```
#9 = Class              #38            // com/tuling/smlz/jvm/classbyatecode/ITulingIntf2
#38 = Utf8              com/tuling/smlz/jvm/classbyatecode/ITulingIntf2
```

第八部分:字段表信息

作用:用于描述类和接口中声明的变量，包括类变量和实例变量

但是不包括方法的局部变量

仅仅的接着接口信息后面的是字段描述 00 01 00 02 00 05 00 06 00 00

00 01 二个字节表示的是field_info字段表的个数 这里很显然只有一个

```
*/
public class TulingByteCode {

    private String userName;

    public String getUserName() { return userName; }



    public void setUserName(String userName) { this.userNa
}
```

字段结构体

Field_info 字段表结构		
类型	名称	数量
u2(1个)	access_flag(权限修饰符)	1
u2	name_index(字段名称索引)	1
u2	descciptor_index(字段描述索引)	1
u2	attribute_count(属性表个数)	1
attribute_info	attributes	attribute_count

00 02 00 05 00 06 00 00

所以00 02 表示访问修饰符号为ACC_PRIVATE

所以00 05 表示的是字段的名称 指向的是常量池中第五个常量

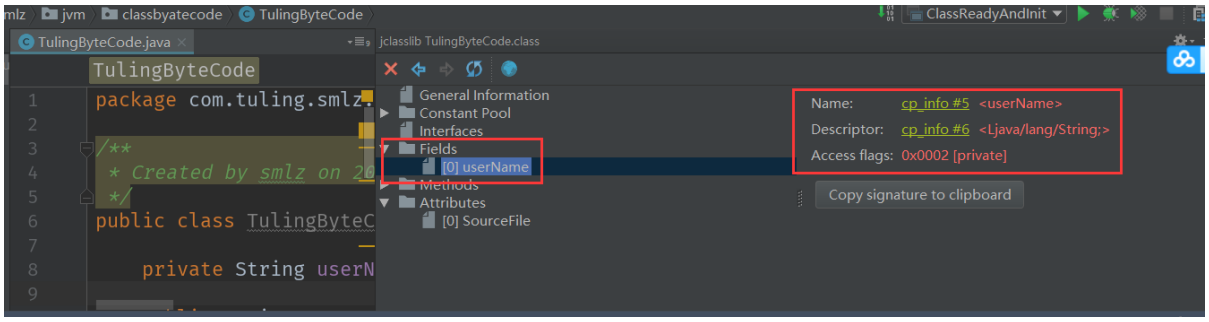
```
#5 = Utf8               userName
```

所以00 06是我们的字段的描述符: 指向的是常量池中第六个常量

```
#6 = Utf8               Ljava/lang/String;
```

00 00 表示是属性表的个数 这里为0表示后面是没有属性表集合

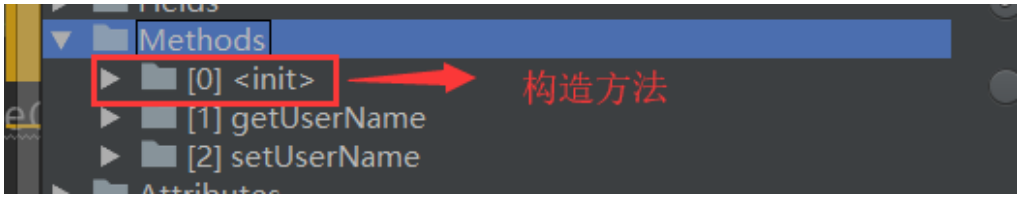
通过jclasslib分析和我们自己分析的出来的结论一致



第九部分:方法表信息分析

```
0  63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43
0  6F 64 65 01 00 10 6A 61  76 61 2F 6C 61 6E 67 2F
0  4F 62 6A 65 63 74 00 21  00 03 00 04 00 00 00 01
0  00 02 00 05 00 06 00 00  00 03 00 01 00 07 00 08
0  00 01 00 09 00 00 00 2F  00 01 00 01 00 00 00 05
0  2A B7 00 01 B1 00 00 00  02 00 0A 00 00 00 06 00
```

00 03 表示我们的方法的个数为三个



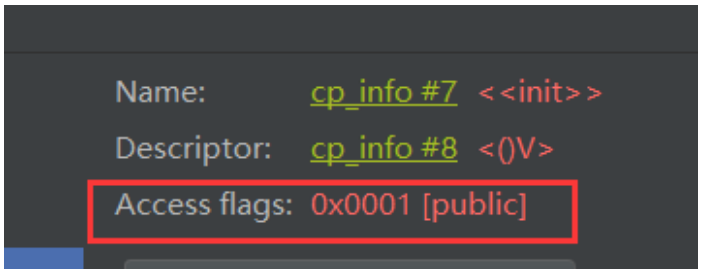
方法表结构:如下

Method_info 字段表结构		
类型	名称	数量
u2	access_flag(权限修饰符)	1
u2	name_index(字段名称索引)	1
u2	descciptor_index(字段描述索引)	1
u2	attribute_count(属性表个数)	1
attribute_info	attributes	attribute_count

第一个方法的前八个字节 00 01 00 07 00 08 00 01

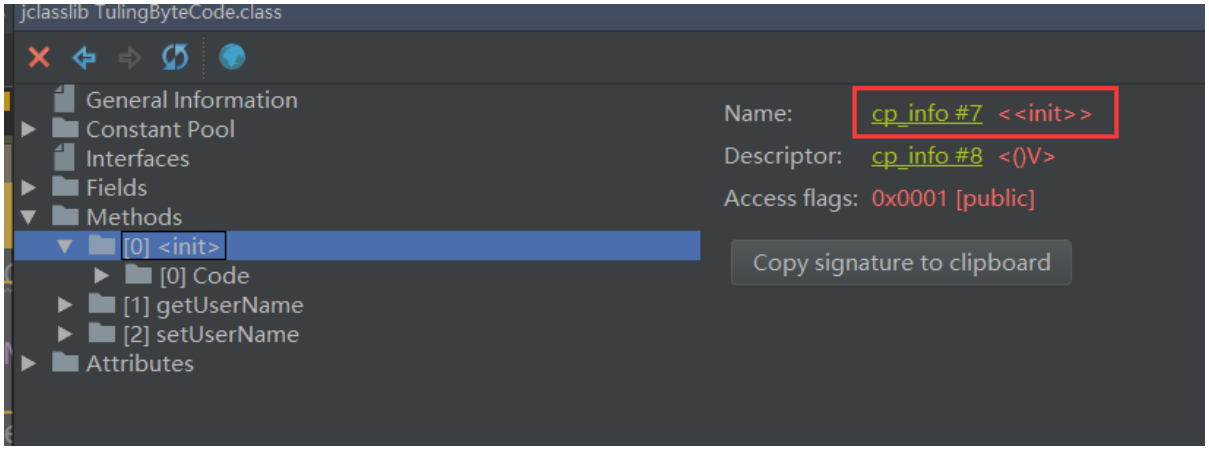
```
L80   4F 62 6A 65 63 74 00 21   00 03 00 04 00 00 00 01   Obj
L90   00 02 00 05 00 06 00 00   00 03 00 01 00 07 00 08   ...
La0   00 01 00 09 00 00 00 2F   00 01 00 01 00 00 00 05   ...
Lb0   2A B7 00 01 B1 00 00 00   02 00 0A 00 00 00 06 00   *..
Lc0   01 00 00 00 06 00 0B 00   00 00 0C 00 01 00 00 00   ...
```

00 01:表示的是方法的修饰符 表示的是acc_public

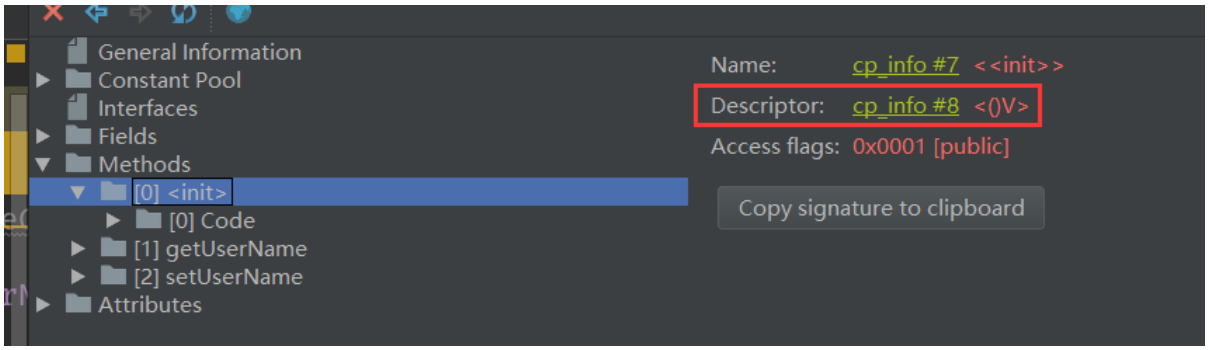


00 07：表示的是方法的名称 表示指向常量池中第7个常量,表示方法的名称

<init>表示构造方法



00 08：方法的描述符号,表示指向常量池第八个常量 为()V 表示的是无参无返回值



00 01表示有一个方法属性的个数

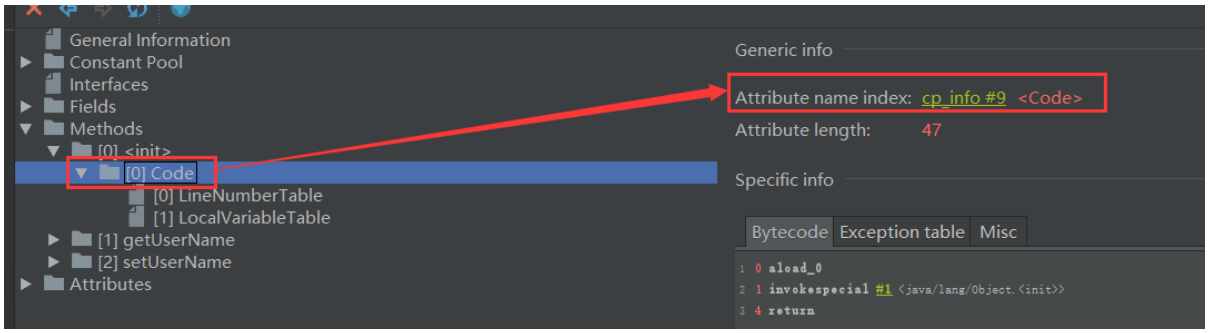
9.1）方法表中的属性表attribute_info结构图

Method_info中attribute_info 结构		
类型	名称	数量
u2	attribute_name_index	1
u4	attribute_length	1
u1	info[attribute_length]	1

```
70  6F 64 65 01 00 10 6A 61  76 61 2F 6C 61 6E 67 2F  ode..
80  4F 62 6A 65 63 74 00 21  00 03 00 04 00 00 00 01  Objec
90  00 02 00 05 00 06 00 00  00 03 00 01 00 07 00 08  .....
a0  00 01 00 09 00 00 00 2F  00 01 00 01 00 00 00 05  ..../...
b0  2A B7 00 01 B1 00 00 00  02 00 0A 00 00 00 06 00  *....
c0  01 00 00 00 06 00 0B 00  00 00 0C 00 01 00 00 00  .....
```

00 09 00 00 00 2F

00 09：表示的是方法属性名称的索引指向常量池#9 表示是Code属性



00 00 00 2F 标识的info的长度 长度值为47个字节 也就是说 会占据47个字节作为code的值

后续的47个字节是我们的Code属性的所占用的字节 (特别特别需要注意这47个字节从Code属性表中第三个开始也就是max_stack开始)

code_attribute属性表结构如下

```
Code_attribute{
    u2          attribute_name_index(2个字节属性名称索引)
    u4          attribute_length(4个字节的属性所包含的字节数，但是不包括attribute_name_index 和attribute_length)
    u2          max_stack;(2个字节，最大操作数栈的深度)
    u2          max_locals;(2个字节,最大的局部变量数,包括传入的入参)
    u4          Code_length;(表示该方法的字节码以及指令码)
    u1          Code[Code_length];(方法指向的具体指令码)
    u2          exception_table_length;(异常表的个数)
    exception_table[exception_table_length];(异常表结构结构，用于存放异常信息)
    {
        u2          start_pc; --start_pc和end_pc表示在code数组中的start_pc和end_pc处(包含start_pc不包含end_pc)指令所
        u2          end_pc;  --抛出的异常由这个表处理
        u2          handler_pc;--表示异常代码的开始处
        u2          catch_type;--表示被处理流程的异常类型，指向常量池中具体的某一个异常类,catchType为0处理所有异常
    }
    u2          attributes_count;
    attribute_info  attributes[ attributes_count];
}
```

```
7A 2F 6A 76 6D 2F 63 6C  61 73 73 62 79 61 74 65  z/jvm/classbyate
63 6F 64 65 2F 54 75 6C  69 6E 67 42 79 74 65 43  code/TulingByteC
6F 64 65 01 00 10 6A 61  76 61 2F 6C 61 6E 67 2F  ode...java/lang/
4F 62 6A 65 63 74 00 21  00 03 00 04 00 00 00 01  Object.!........
00 02 00 05 00 06 00 00  00 03 00 01 00 07 00 08  ................
00 01 00 09 00 00 00 2F  00 01 00 01 00 00 00 05  ......./........
2A B7 00 01 B1 00 00 00  02 00 0A 00 00 00 06 00  *...............
01 00 00 00 06 00 0B 00  00 00 0C 00 01 00 00 00  ................
05 00 0C 00 0D 00 00 00  01 00 0E 00 0F 00 01 00  ................
09 00 00 00 2F 00 01 00  01 00 00 00 05 2A B4 00  ..../........*..
02 B0 00 00 00 02 00 0A  00 00 00 06 00 01 00 00  ................
```

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00 00

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00 00

(max_stack)表示的是我们最大操作数栈的深度为1

```
public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
  descriptor: ()V
  flags: ACC_PUBLIC
  Code:
    stack=1, locals=1, args_size=1
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return
    LineNumberTable:
      line 6: 0
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

(max_locals)标识的是局部变量表变量的个数

```
public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
  descriptor: ()V
  flags: ACC_PUBLIC
  Code:
    stack=1, locals=1, args_size=1
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return
    LineNumberTable:
      line 6: 0
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

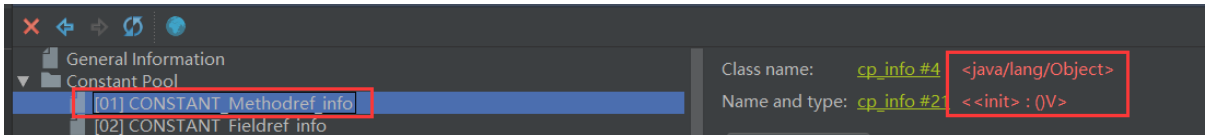
00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

(Code_lenth)四个字节表示的是我们指令的长度为五

字节码指令助记符号　2A B7 00 01 B1

0x2A:对应的字节码注记符是aload_0,作用就是把当前调用方法的栈帧中的局部变量表索引位置为0的局部变量推送到操作数栈的栈顶.

0xB7:表示是 invokespecial 调用父类的方法 那么后面需要接入二个字节表示调用哪个方法 所以 00 01 表示的是指向常量池中第一个位置为为如下结构



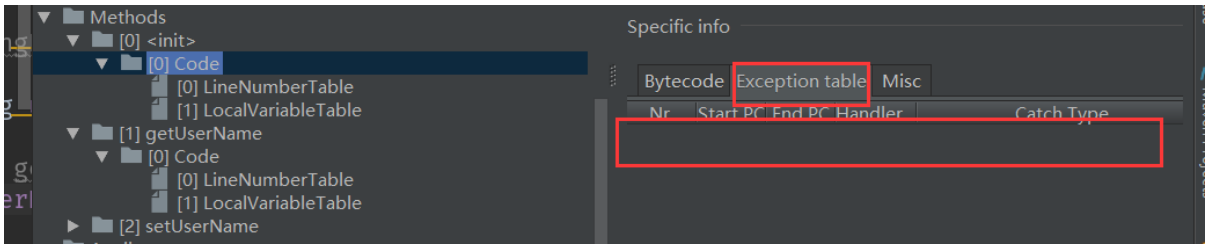

```
public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
  descriptor: ()V
  flags: ACC_PUBLIC
  Code:
    stack=1, locals=1, args_size=1
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return
    LineNumberTable:
      line 6: 0
    LocalVariableTable:
```

0xB1 对应的字节码指令值retrun　表示retrun void from method;

```
public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
  descriptor: ()V
  flags: ACC_PUBLIC
  Code:
    stack=1, locals=1, args_size=1
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return
    LineNumberTable:
      line 6: 0
    LocalVariableTable:
```

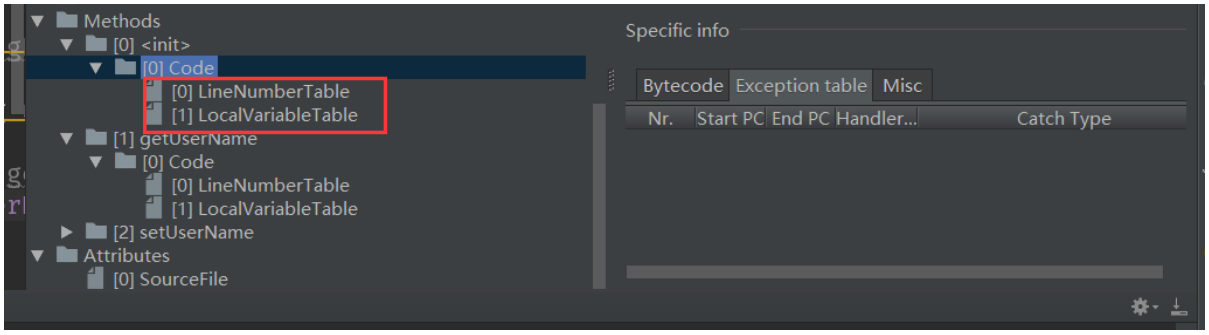
异常信息表的个数为 00 00 表示方法没有抛出异常

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00



表示Code_attribute结构中属性表的个数为00 02 表示为2个

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

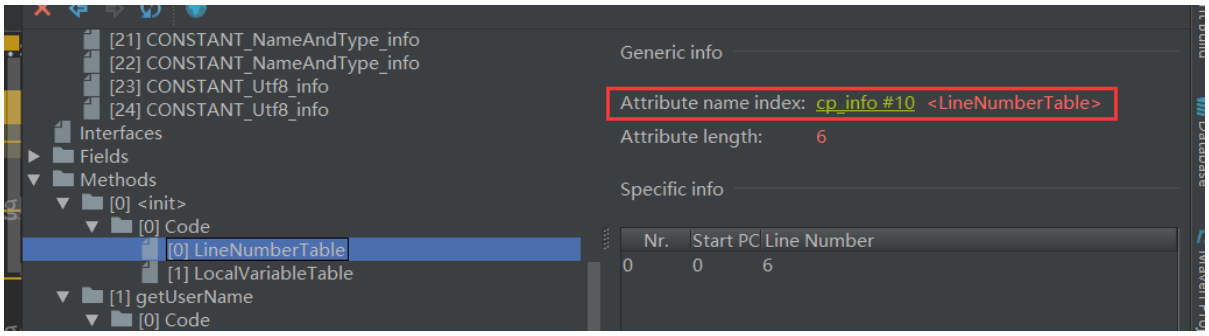


LineNumberTable结构体为下图

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

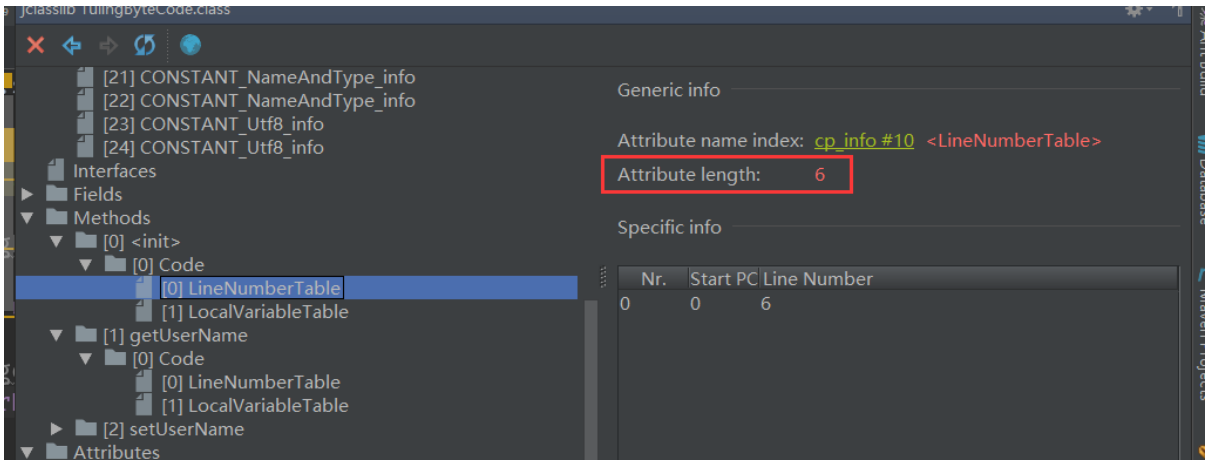
00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

这二个字节表示的是我们属性名称的索引attribute_name_index指向常量池中的00 0A #10个常量池



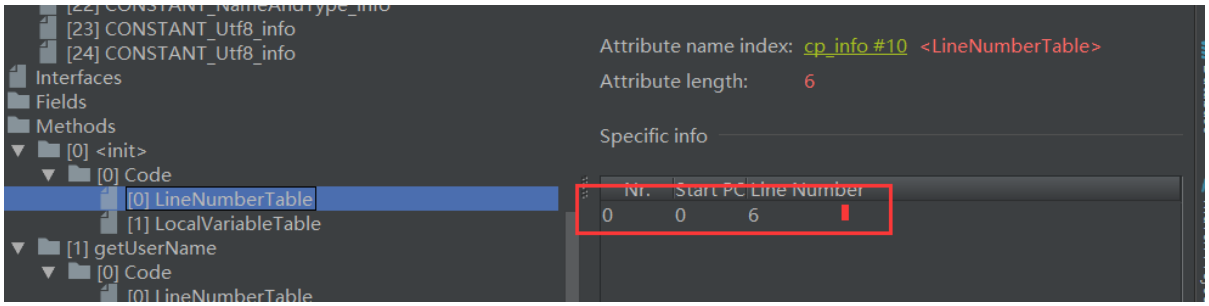
attribute_length:属性的长度占用四个字节: 表示后面00 00 00 06 六个字节是我们属性的内容

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00



这里的00 01 表示的是有几对指令码和源码映射关系 这里明显只有一对

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00



00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

这里表示 第一个指令码映射的是第六行源码

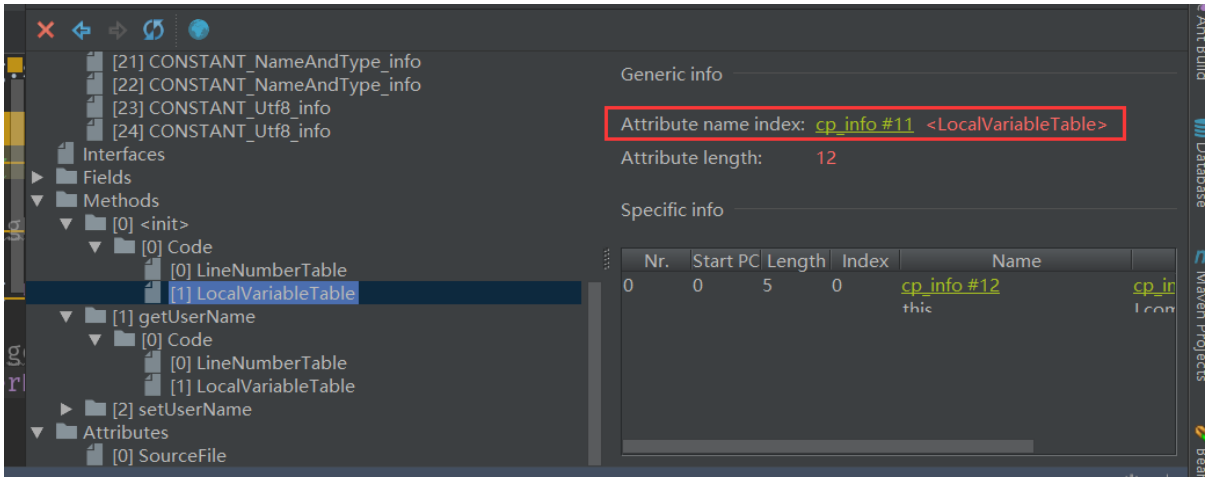
LocalVariableTable 本地方法变量表结构分析

表 6-19 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

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

本地变量表的名称的索引指向attribute_name_index的是常量池11的位置:



本地变量表中属性的长度attribute_length:12长度

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00

00

本地变量表local_variable_table_length的个数

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

local_vabiable_info的结构

表 6-20　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

start_pc:这个局部变量的生命周期开始的字节码偏移量 占用二个字节

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
{
  public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1                  // Method java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 6: 0
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

length:作用范围覆盖的长度 占用二个字节

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
#24 = Utf8               java/lang/Object
{
  public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1                  // Method java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 6: 0
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

name_index:表示局部变量的名称 二个字节

00 0C表示指向常量池12的位置

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
#12 = Utf8               this
```

```
{
  public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1                  // Method java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 6: 0
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

desc_index:表示局部变量描述符索引 二个字节

00 0D表示指向常量池13的位置

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
  #14 = Utf8               getUserName
```

```
{
  public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1                  // Method java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 6: 0
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

index:index是这个局部变量在栈帧局部变量表中Slot的位置。当这个变量数据类型是64位类型时（double和long），它占用的Slot为index和index+1两个

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
  #24 = Utf8               java/lang/Object
{
  public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1                  // Method java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 6: 0
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

第二个方法的字节码

```
public java.lang.String getUserName();
  descriptor: ()Ljava/lang/String;
  flags: ACC_PUBLIC
  Code:
    stack=1, locals=1, args_size=1
       0: aload_0
       1: getfield      #2                  // Field userName:Ljava/lang/String;
       4: areturn
    LineNumberTable:
      line 11: 0
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

1)方法访问标记符号:

0x0001我们根据访问权限修饰符号查询可得 访问权限是ACC_PUBLIC

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

2）方法名称: 00 0E指向常量池中#14的位置

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
#14 = Utf8               getUserName
```

3)方法描述符号:00 0F 指向我们的常量池#15的位置

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00

00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
#15 = Utf8               ()Ljava/lang/String;
```

4)方法表属性个数 00 01 表示一个

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

5)方法结构体中的attribute_info的结构体

Method_info中attribute_info 结构		
类型	名称	数量
u2	attribute_name_index	1
u4	attribute_length	1
u1	info[attribute_length]	1

5.1)attribute_info.attribute_name_index 表示的数属性名称索引 0009指向常量池的位置: Code

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
#9 = Utf8               Code
```

5.2)attribute_info.attribute_length表示的是我们属性的长度 00 00 00 2F

表示后面47个字节都是我们的Code_info结构体

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00

00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

5.3）Code_info结构体如图所示

```
Code_attribute{
    u2              attribute_name_index(2个字节属性名称索引)
    u4              attribute_length(4个字节的属性所包含的字节数，但是不包括attribute_name_index 和attribute_length)
    u2              max_stack;(2个字节，最大操作数栈的深度)
    u2              max_locals;(2个字节,最大的局部变量数,包括传入的入参)
    u4              Code_length;(表示该方法的字节码以及指令码)
    u1              Code[Code_length];(方法指向的具体指令码)
    u2              exception_table_length;(异常表的个数)
    exception_table[exception_table_length];(异常表结构结构，用于存放异常信息)
    {
        u2          start_pc; --start_pc和end_pc表示在code数组中的start_pc和end_pc处(包含start_pc不包含end_pc)指令所
        u2          end_pc;  --抛出的异常由这个表处理
        u2          handler_pc;--表示异常代码的开始处
        u2          catch_type;--表示被处理流程的异常类型，指向常量池中具体的某一个异常类,catchType为0处理所有异常
    }
    u2              attributes_count;
    attribute_info  attributes[ attributes_count];
}
```

5.3.1)Code_info.max_stack方法操作数栈的深度

00 01表示方法操作数栈的深度为1

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
public java.lang.String getUserName();
  descriptor: ()Ljava/lang/String;
  flags: ACC_PUBLIC
  Code:
    stack=1, locals=1, args_size=1
       0: aload_0
       1: getfield      #2                  // Field userName:Ljava/lang/String;
       4: areturn
    LineNumberTable:
      line 11: 0
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

5.3.2)Code_info.max_locals方法局部变量表的个数

00 01方法局部变量表的个数 1

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
public java.lang.String getUserName();
  descriptor: ()Ljava/lang/String;
  flags: ACC_PUBLIC
  Code:
    stack=1, locals=1, args_size=1
       0: aload_0
       1: getfield      #2                  // Field userName:Ljava/lang/String;
       4: areturn
    LineNumberTable:
      line 11: 0
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

5.3.3)Code_info.code_length 指令码的长度 00 00 00 05 后面紧接着5个字节就是我们的具体指令码

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

5.3.4)Code_info.code[code_length] 表示后面五个字节就是我们的指令码

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

①:0x2A:对应的字节码注记符是aload_0,作用就是把当前调用方法的栈帧中的局部变量表索引位置为0的局部变量推送到操作数栈的栈顶.

②:0xb4 getfield 获取指定类的实例域，并将其值压入栈顶 后面是操作的字段

00 02表示常量池索引第二位置

B4 00 02 表示的意思就是把userName类型实例变量的引用压入方法的操作数栈

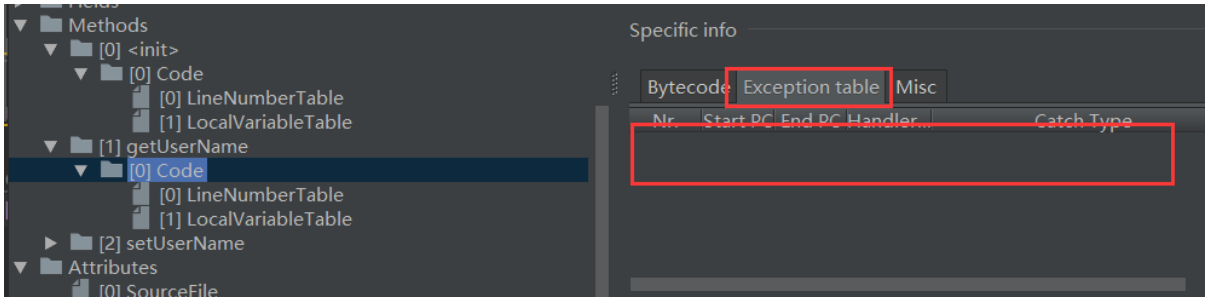
```
Constant pool:
   #1 = Methodref          #4.#21         // java/lang/Object."<init>":()V
   #2 = Fieldref           #3.#22         // com/tuling/smlz/jvm/classbyatecode/TulingByteCode.userName:Ljava/lang/String;
   #3 = Class              #23            // com/tuling/smlz/jvm/classbyatecode/TulingByteCode
   #4 = Class              #24            // java/lang/Object
```

```
public java.lang.String getUserName();
  descriptor: ()Ljava/lang/String;
  flags: ACC_PUBLIC
  Code:
    stack=1, locals=1, args_size=1
       0: aload_0
       1: getfield      #2                  // Field userName:Ljava/lang/String;
       4: areturn
    LineNumberTable:
      line 11: 0
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

③:0xB4---->表示为aretrun 返回 从当前方法返回对象引用

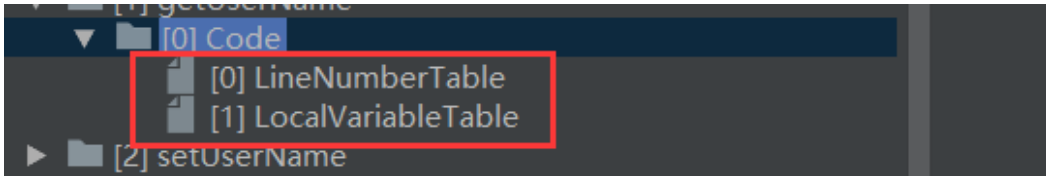
5.3.5）Code_info.exception_table_length 异常表的个数: 00 00表示方法没有异常

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00



5.3.6)code_info.attribute_count 表示code_info属性attribute_info的个数 2个

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00



5.3.7)code_info.attribute_info[1]

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

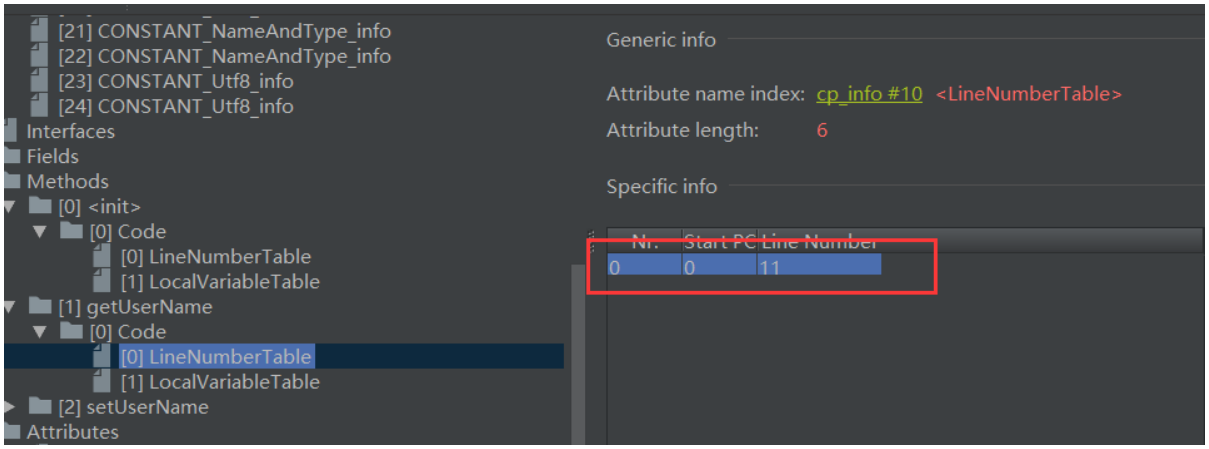
①:00 0A表示为attribute_name_index指向常量池10的位置

```
#10 = Utf8               LineNumberTable
```

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

②: 00 00 00 06 表示的是attribute_length 表示长度,接着后面6个字节是我们的line_number_info的结构体所再用的字节

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00



③:line_number_info结构体

00 01：表示字节码和源码映射的对数　01表示一对

00 00: 方法中的字节码的行号

00 0B:　源码中的行号 11行

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

5.3.7)code_info.attribute_info[2]

表 6-19　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

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

attribute_name_index 00 0B 表示的是指向我们的常量池中11的位置为LocalVariableTable

```
#11 = Utf8               LocalVariableTable
```

attribute_length:00 00 00 0C 标识属性的长度为12，那么后面的12个字符就是我们的属性表的内容

local_varibale_info表的结构体

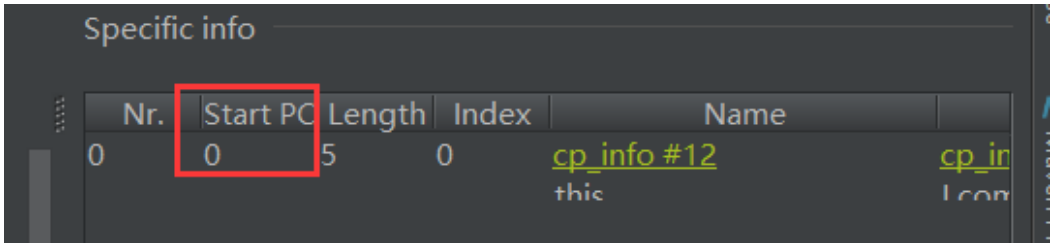
表 6-20　local_variable_info 项目结构

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

00 01 00 0E 00 0F 00 01 00 09 00 00 00 2F 00 01 00 01 00 00 00 05 2A B4 00 02 B0 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 0B 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

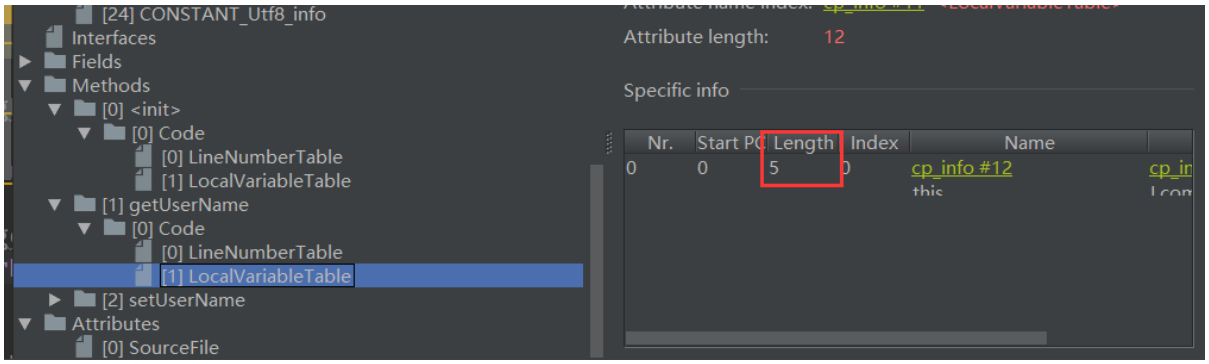
start_pc:这个局部变量的生命周期开始的字节码偏移量 占用二个字节

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00



length:作用范围覆盖的长度 占用二个字节

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00



name_index:表示局部变量的名称 二个字节

00 0C表示指向常量池12的位置

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

desc_index:表示局部变量描述符索引 二个字节

00 0D表示指向常量池13的位置

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

index:index是这个局部变量在栈帧局部变量表中Slot的位置。当这个变量数据类型是64位类型时（double和long），它占用的Slot为index和index+1两个

00 01 00 01 00 00 00 05 2A B7 00 01 B1 00 00 00 02 00 0A 00 00 00 06 00 01 00 00 00 06 00 0B 00 00 00 0C 00 01 00 00 00 05 00 0C 00 0D 00 00

```
  #24 = Utf8               java/lang/Object
{
  public com.tuling.smlz.jvm.classbyatecode.TulingByteCode();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1                  // Method java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 6: 0
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       5     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
```

第三个方法的字节码文件分析

```
  public void setUserName(java.lang.String);
    descriptor: (Ljava/lang/String;)V
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=2, args_size=2
         0: aload_0
         1: aload_1
         2: putfield      #2                  // Field userName:Ljava/lang/String;
         5: return
      LineNumberTable:
        line 15: 0
        line 16: 5
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       6     0  this   Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;
            0       6     1 userName   Ljava/lang/String;
    MethodParameters:
      Name                           Flags
      userName
}
```

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

1)访问权限修饰符(这个权限修饰符为0x0001)

那么权限符是acc_public

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

2）方法名称索引指向常量池中的#16

#16 = Utf8 setUserName

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

3)方法描述符号索引 指向常量池中#17的位置

#17 = Utf8 (Ljava/lang/String;)V

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

4)该方法属性表的个数 为2个

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

4.1)第一个属性表Code属性表结构

表 6-15　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

①: 属性名称索引 指向常量池第九个位置

#9 = Utf8　　Code

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

②：属性长度 占用四个字节，四个字节计算出来的字节数字就是我们的Code属性内容

00 00 00 3E 转换成62个字节，那么我们后面的62个字节是我们的属性内容

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

③：本方法最大操作数深度为2

```
public void setUserName(java.lang.String);
  descriptor: (Ljava/lang/String;)V
  flags: ACC_PUBLIC
  Code:
    stack=2, locals=2, args_size=2
       0: aload_0
       1: aload_1
       2: putfield      #2                  // Field userName:Ljava/lang/String;
       5: return
     LineNumberTable:
```

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

④:局部变量表的大小为2

```
public void setUserName(java.lang.String);
  descriptor: (Ljava/lang/String;)V
  flags: ACC_PUBLIC
  Code:
    stack=2, locals=2, args_size=2
       0: aload_0
       1: aload_1
       2: putfield      #2                  // Field userName:Ljava/lang/String;
       5: return
     LineNumberTable:
```

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

⑤：jvm指令码长度,占用四个字节 00 00 00 06

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

⑥：jvm 指令解析　6个字节的指令码2A 2B B5 00 02 B1

```
0: aload_0
1: aload_1
2: putfield      #2                  // Field userName:Ljava/lang/String;
5: return
```

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

a) 0x2A->aload_0：表示把引用类型的压到我们操作数栈栈顶

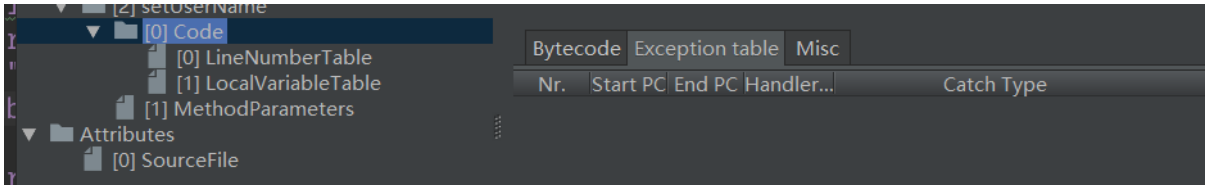
b) 0x2B->aload_1：把我们第二个引用类型压入到操作数栈顶

c)0xB5->putFiled 把我们的栈顶的值赋值给实例变量

d)00 02: 表示putFiled的字端，表示操作的对象 指向我们的常量池#2的位置

e)0xB1:->从当前方法返回void

⑦:exception_table_length 异常表长度 为0，那么异常表个数为0



00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

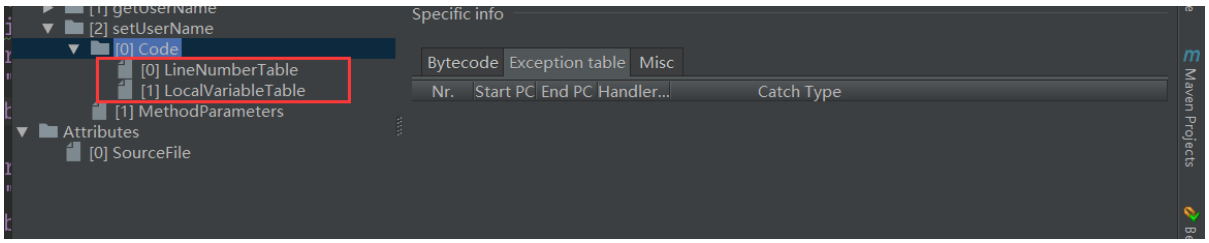
00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

⑧:Code属性表的attribute_info_count Code属性表的attribute属性个数



00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

A)Code_info的第一个属性表之lineNumberTable

表 6-18　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

attribute_name_index:0A 指向我们的常量池10的位置

#10 = Utf8   LineNumberTable

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

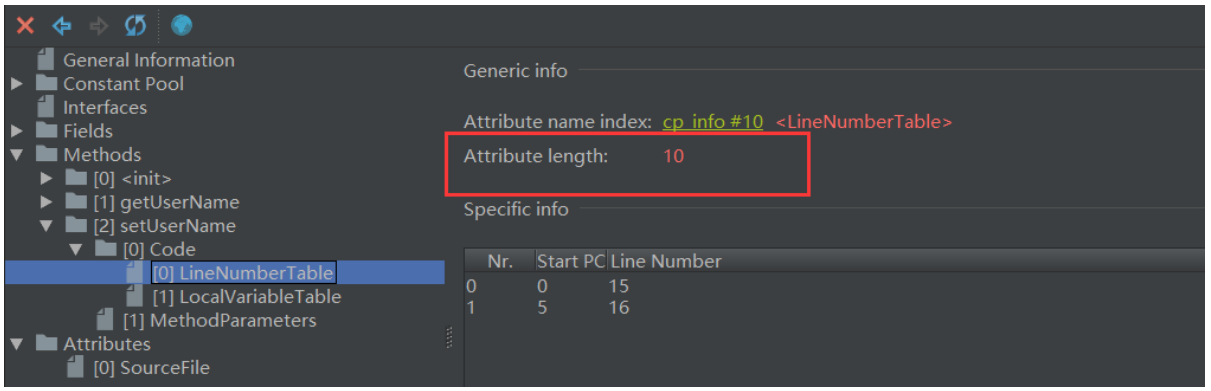
00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

attribute_length:占用四个字节 00 00 00 0A(10字节)

表示后面字节是我们的具体的属性



00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

line_number_table_length:占用二个字节 表示2对映射

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

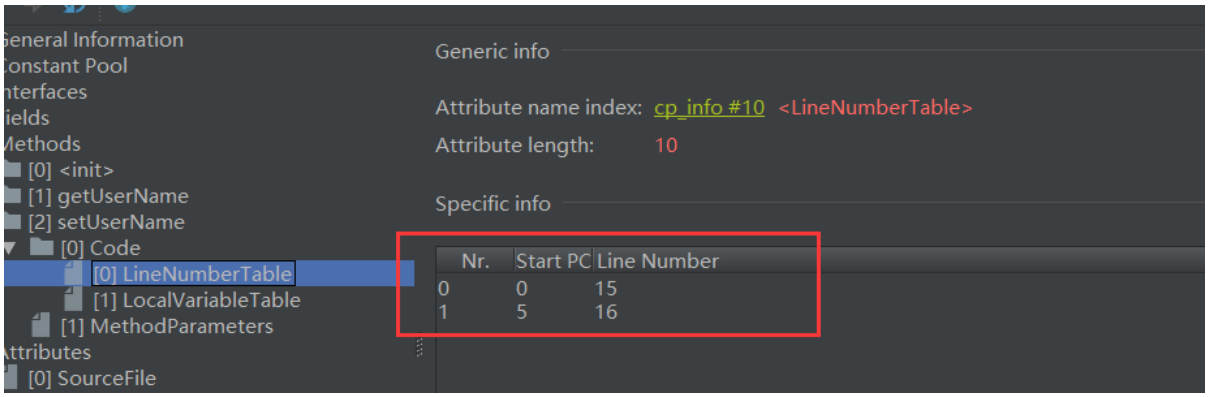
00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

line_number_info存在二对映射



00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

指令码 00 映射源码 00 15(21)行

指令码05 映射源码00 16(22行)

B)Code_info的第二个属性表之LocalVariableTable

表 6-19　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

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

attribute_name_index 00 0B指向我们的常量池11的位置

#11 = Utf8　　LocalVariableTable

attribute_length表示属性的长度,后面的22个字节都是我们的属性类容

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

local_variable_table_length 00 02(表示有二个本地变量表)

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

Local_variale_info的变量表结构

表 6-20　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

第一个变量表:

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

"start_pc":"u2(00 00)->desc:这个局部变量的生命周期开始的字节码偏移量",

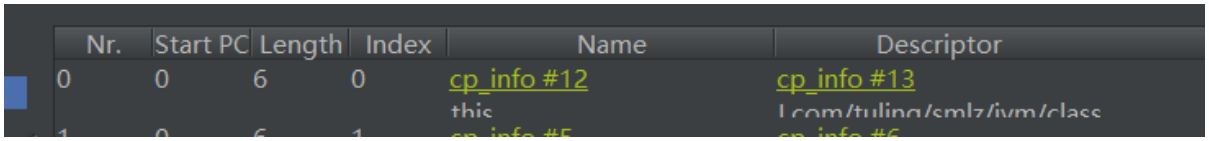
"length:":"u2(00 06)->作用范围覆盖的长度为6",

"name_index":"u2(00 0c)->字段的名称索引指向常量池12的位置 this",

"desc_index":"u2(00 0D)局部变量的描述符号索引->指向#13的位置

Lcom/tuling/smlz/jvm/classbyatecode/TulingByteCode;",

"index":"u2(00 00)->desc:index是这个局部变量在栈帧局部变量表中Slot的位置"



第二个变量表:

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

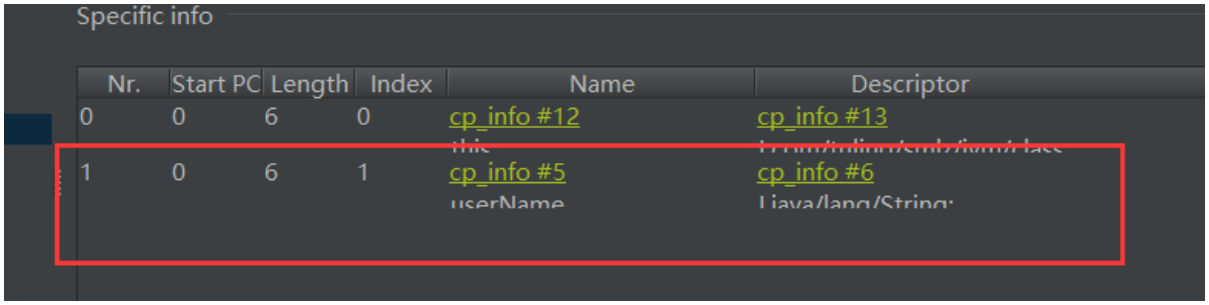
"start_pc":"u2(00 00)->desc:这个局部变量的生命周期开始的字节码偏移量",

"length:":"u2(00 06)->作用范围覆盖的长度为6",

"name_index":"u2(00 05)->字段的名称索引指向常量池5的位置 userName",

"desc_index":"u2(00 06)局部变量的描述符号索引->指向#6的位置 Ljava/lang/String;",

"index":"u2(00 01)->desc:index是这个局部变量在栈帧局部变量表中Slot的1位置"



B)Code_info的第二个属性表之MethodParameters方法参数属性表

00 01 00 10 00 11 00 02 00 09 00 00

00 3E 00 02 00 02 00 00 00 06 2A 2B B5 00 02 B1

00 00 00 02 00 0A 00 00 00 0A 00 02 00 00 00 15

00 05 00 16 00 0B 00 00 00 16 00 02 00 00 00 06

00 0C 00 0D 00 00 00 00 00 06 00 05 00 06 00 01

00 12 00 00 00 05 01 00 05 00 00

结构体:

```
{
  "attribute_name_index":"u2(00 12)表示该属性的名称指向常量池#18的位置:MethodParameters",
  "attribute_length":"u4(00 00 00 05 )->desc:属性的长度5",
  "parameter_count":"u1(01)->desc参数的个数1个",
  "parameter_name_index":"u2(00 05)->desc:指向第五个常量池的常量userName",
  "ACC_FLAG":"U2(00 00 )->desc:表示任何权限都可以访问"
}
```

最后一部分:class文件的属性

00 01 00 13 00 00 00 02 00 14

"attribute_count(class文件的属性)":"u2(00 01)只有一个属性"

属性结构体:

```
{
 "attribute_name_index":"u2(00 13) 指向常量池中#19 值为 SourceFile",
 "attribute_length":"u4(00 00 00 02) 表示属性接下来的长度为2",
 "sourceFile_index":"u2(00 14) 表示源文件的索引指向常量池20的位置:TulingByteCode.java"
}
```

有道云 文档：第二节 class文件结构(简单class文件)....
链接：http://note.youdao.com/noteshare?id=a21875fd4ab6e14a445d07e750cc6930&sub=30E443956A304A669821CFAB0ACE45EB