https://vinoit.me/2016/05/22/java-classfile-structure-detail/

本文描述的是class结构的具体信息，排版顺序按照class结构。

常量池

通用格式

所有的常量池项都具有如下通用格式：

```
cp_info {
    u1 tag;
    u1 info[];
}
```

常量池的 tag 项说明

常量类型	值
CONSTANT_Class	7
CONSTANT_Fieldref	9
CONSTANT_Methodref	10
CONSTANT_InterfaceMethodref	11
CONSTANT_String	8
CONSTANT_Integer	3
CONSTANT_Float	4
CONSTANT_Long	5
CONSTANT_Double	6
CONSTANT_NameAndType	12
CONSTANT_Utf8	1
CONSTANT_MethodHandle	15
CONSTANT_MethodType	16
CONSTANT_InvokeDynamic	18

1. CONSTANT_Class_info 结构

CONSTANT_Class_info 结构用于表示类或接口，格式如下：

```
CONSTANT_Class_info {
    u1 tag;
    u2 name_index;
}
```

2. CONSTANT_Fieldref_info 结构

```
CONSTANT_Fieldref_info {
```

```
    u1 tag;
    u2 class_index;
    u2 name_and_type_index;
}
```

3. CONSTANT_Methodref_info 结构

```
CONSTANT_Methodref_info {
    u1 tag;
    u2 class_index;
    u2 name_and_type_index;
}
```

4. CONSTANT_InterfaceMethodref_info 结构

```
CONSTANT_InterfaceMethodref_info {
    u1 tag;
    u2 class_index;
    u2 name_and_type_index;
}
```

5. CONSTANT_String_info 结构

CONSTANT_String_info 用于表示 java.lang.String 类型的常量对象，格式如下：

```
CONSTANT_String_info {
    u1 tag;
    u2 string_index;
}
```

6. CONSTANT_Integer_info 结构

```
CONSTANT_Integer_info {
    u1 tag;
    u4 bytes;
}
```

7. CONSTANT_Float_info 结构

```
CONSTANT_Float_info {
    u1 tag;
    u4 bytes;
}
```

8. CONSTANT_Long_info 结构

```
CONSTANT_Long_info {
```

```
    u1 tag;
    u4 high_bytes;
    u4 low_bytes;
}
```

9. CONSTANT_Double_info 结构

```
CONSTANT_Double_info {
    u1 tag;
    u4 high_bytes;
    u4 low_bytes;
}
```

10. CONSTANT_NameAndType_info 结构

CONSTANT_NameAndType_info 结构用于表示字段或方法

```
CONSTANT_NameAndType_info {
    u1 tag;
    u2 name_index; //name_index 项的值必须是对常量池的有效索引， 常量池在该索引处的项必须是
    CONSTANT_Utf8_info（ §4.4.7）结构，这个结构要么表示特殊的方法名<init>，要么表示一个有效
    的字段或方法的非限定名（ Unqualified Name）。
    u2 descriptor_index;//descriptor_index 项的值必须是对常量池的有效索引， 常量池在该索引
    处的项必须是CONSTANT_Utf8_info结构。
}
```

方法描述符具体格式可参考《JVM规范》。

11. CONSTANT_Utf8_info 结构

CONSTANT_Utf8_info 结构用于表示字符串常量的值：

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

12. CONSTANT_MethodHandle_info 结构

CONSTANT_MethodHandle_info 结构用于表示方法句柄，结构如下：

```
CONSTANT_MethodHandle_info {
    u1 tag;
    u1 reference_kind;//reference_kind 项的值必须在 1 至 9 之间（包括 1 和 9），它决
```

定了方法句柄的类型。方法句柄类型的值表示方法句柄的字节码行为。
    u2 reference_index;//reference_index 项的值必须是对常量池的有效索引。
}

13. CONSTANT_MethodType_info 结构

CONSTANT_MethodType_info 结构用于表示方法类型：

```
CONSTANT_MethodType_info {
    u1 tag;
    u2 descriptor_index;
}
```

14. CONSTANT_InvokeDynamic_info 结构

CONSTANT_InvokeDynamic_info 用于表示 invokedynamic 指令所使用到的引导方法
（ Bootstrap Method ）、 引导方法使用到动态调用名称（ Dynamic Invocation Name ）、 参
数和请求返回类型、以及可以选择性的附加被称为静态参数（ Static Arguments ）的常量序列。

```
CONSTANT_InvokeDynamic_info {
    u1 tag;
    u2 bootstrap_method_attr_index;
    u2 name_and_type_index;
}
```

access_flags的取值范围和相应含义表

标记名	值	含义
ACC_PUBLIC	0x0001	可以被包的类外访问。
ACC_FINAL	0x0010	不允许有子类。
ACC_SUPER	0x0020	当用到 invokespecial 指令时，需要特殊处理的父类方法。
ACC_INTERFACE	0x0200	标识定义的是接口而不是类。
ACC_ABSTRACT	0x0400	不能被实例化。
ACC_SYNTHETIC	0x1000	标识并非 Java 源码生成的代码。
ACC_ANNOTATION	0x2000	标识注解类型
ACC_ENUM	0x4000	标识枚举类型

字段

field_info 结构格式如下：

```
field_info {
    u2 access_flags;
    u2 name_index;
    u2 descriptor_index;
    u2 attributes_count;
    attribute_info attributes[attributes_count];
}
```

字段 access_flags 标记列表及其含义

标记名	值	说明
ACC_PUBLIC	0x0001	public，表示字段可以从任何包访问。
ACC_PRIVATE	0x0002	private，表示字段仅能该类自身调用。
ACC_PROTECTED	0x0004	protected，表示字段可以被子类调用。
ACC_STATIC	0x0008	static，表示静态字段。
ACC_FINAL	0x0010	final，表示字段定义后值无法修改
ACC_VOLATILE	0x0040	volatile，表示字段是易变的。
ACC_TRANSIENT	0x0080	transient，表示字段不会被序列
ACC_SYNTHETIC	0x1000	表示字段由编译器自动产生。
ACC_ENUM	0x4000	enum，表示字段为枚举类型

基本类型字符解释表

字符	类型	含义
B	byte	有符号字节型数

方法 access_flags 标记列表及其含义

标记名	值	说明
ACC_PUBLIC	0x0001	public，方法可以从包外访问
ACC_PRIVATE	0x0002	private，方法只能本类中访问
ACC_PROTECTED	0x0004	protected，方法在自身和子类可以访问
ACC_STATIC	0x0008	static，静态方法
ACC_FINAL	0x0010	final，方法不能被重写（覆盖）
ACC_SYNCHRONIZED	0x0020	synchronized，方法由管程同步
ACC_BRIDGE	0x0040	bridge，方法由编译器产生
ACC_VARARGS	0x0080	表示方法带有变长参数
ACC_NATIVE	0x0100	native，方法引用非 java 语言的本地方法
ACC_ABSTRACT	0x0400	abstract，方法没有具体实现
ACC_STRICT	0x0800	strictfp，方法使用 FP-strict 浮点格式
ACC_SYNTHETIC	0x1000	方法在源文件中不出现，由编译器产生

属性

属性表的限制相对宽松，不需要各个属性表有严格的顺序，只有不与已有的属性名重复，任何自定义的编译器都可以向属性表中写入自定义的属性信息，Java虚拟机运行时会忽略掉无法识别的属性。

关于虚拟机规范中预定义的属性，这里不展开讲了，列举几个常用的。

属性的通用格式如下：

```
attribute_info {
    u2 attribute_name_index;    //属性名索引
    u4 attribute_length;        //属性长度
    u1 info[attribute_length]; //属性的具体内容
}
```

ConstantValue 属性

ConstantValue 属性表示一个常量字段的值。位于 field_info结构的属性表中。

```
ConstantValue_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 constantvalue_index;//字段值在常量池中的索引，常量池在该索引处的项给出该属性表示的常量值。（例如，值是long型的，在常量池中便是CONSTANT_Long）
}
```

Deprecated 属性

Deprecated 属性是在 JDK 1.1 为了支持注释中的关键词@deprecated 而引入的。

Deprecated 属性格式如下：

```
Deprecated_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
}
```

Code 属性

Code 属性的格式如下：

```
Code_attribute {
    u2 attribute_name_index; //常量池中的uft8类型的索引，值固定为”Code“
    u4 attribute_length; //属性值长度，为整个属性表长度-6
    u2 max_stack;  //操作数栈的最大深度值，jvm运行时根据该值分配栈帧
    u2 max_locals;  //局部变量表最大存储空间，单位是slot
    u4 code_length; // 字节码指令的个数
    u1 code[code_length]; // 具体的字节码指令
    u2 exception_table_length; //异常的个数
    {   u2 start_pc;
        u2 end_pc;
        u2 handler_pc; //当字节码在[start_pc, end_pc)区间出现catch_type或子类，则转到handler_pc行继续处理。
        u2 catch_type; //当catch_type=0，则任意异常都需转到handler_pc处理
    } exception_table[exception_table_length]; //具体的异常内容
    u2 attributes_count;     //属性的个数
    attribute_info attributes[attributes_count]; //具体的属性内容
}
```

- 其中slot为局部变量中的最小单位。boolean、 byte、 char、 short、 float、 reference和 returnAddress 等小于等于32位的用一个slot表示，double,long这些大于32位的用2个slot表示

InnerClasses 属性

为了方便说明特别定义一个表示类或接口的 Class 格式为 C。如果 C 的常量池中包含

某个
CONSTANT_Class_info 成员，且这个成员所表示的类或接口不属于任何一个包，那么 C 的
ClassFile 结构的属性表中就必须含有对应的 InnerClasses 属性。
InnerClasses 属性是在 JDK 1.1 中为了支持内部类和内部接口而引入的,位于 ClassFile结构的属性表。

```
InnerClasses_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 number_of_classes;
    {   u2 inner_class_info_index;
        u2 outer_class_info_index;
        u2 inner_name_index;
        u2 inner_class_access_flags;
    } classes[number_of_classes];
}
```

LineNumberTable 属性

LineNumberTable 属性是可选变长属性，位于 Code结构的属性表。它被调试器用于确定源文件中行号表示的内容在 Java 虚拟机的 code[]数组中对应的部分。在 Code 属性
的属性表中， LineNumberTable 属性可以按照任意顺序出现，此外，多个 LineNumberTable
属性可以共同表示一个行号在源文件中表示的内容，即 LineNumberTable 属性不需要与源文件
的行一一对应。
LineNumberTable 属性格式如下：

```
LineNumberTable_attribute {
    u2 attribute_name_index;//属性名称在常量池的索引，指向一个 CONSTANT_Utf8_info结构。
    u4 attribute_length;//属性长度
    u2 line_number_table_length;//线性表长度
    {   u2 start_pc;
        u2 line_number;
      } line_number_table[line_number_table_length];
}
```

LocalVariableTable 属性

LocalVariableTable 是可选变长属性，位于 Code属性的属性表中。它被调试器用于确定方法在执行过程中局部变量的信息。在 Code 属性的属性表中，

LocalVariableTable 属性可以按照任意顺序出现。 Code 属性中的每个局部变量最多只能有一
个 LocalVariableTable 属性。
LocalVariableTable 属性格式如下：

```
LocalVariableTable_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 local_variable_table_length
    { u2 start_pc;
      u2 length;
      u2 name_index;
      u2 descriptor_index;
      u2 index;
    } local_variable_table[local_variable_table_length];
}
```

Signature 属性

Signature 属性是可选的定长属性，位于 ClassFile， field_info 或 method_info结构的属性表中。在 Java 语言中，任何类、 接口、 初始化方法或成
员的泛型签名如果包含了类型变量（ Type Variables ）或参数化类型（ Parameterized
Types ），则 Signature 属性会为它记录泛型签名信息。
Signature 属性格式如下：

```
Signature_attribute {
    u2 attribute_name_index;//属性名称在常量池中的索引，指向一个CONSTANT_Utf8_info结构。
    u4 attribute_length;
    u2 signature_index;
}
```

- slot是虚拟机未局部变量分配内存使用的最小单位。对于byte/char/float/int/short/boolean/returnAddress等长度不超过32位的局部变量，每个占用1个Slot；对于long和double这两种64位的数据类型则需要2个Slot来存放。
- 实例方法中有隐藏参数this, 显式异常处理器的参数，方法体定义的局部变量都使用局部变量表来存放。
- max_locals，不是所有局部变量所占Slot之和，因为Slot可以重用，javac编译器会根据变量的作用域来分配Slot给各个变量使用，从而计算出max_locals大小。

- 虚拟机规范限制严格方法不允许超过65535个字节码，否则拒绝编译。
- 参考《JVM规范SE7》