**规则结构类型列表：**

|  |  |  |
| --- | --- | --- |
| **Type** | **Meaning** | **Used For** |
| 0 | Varint | int32, int64, uint32, uint64, sint32, sint64, bool, enum |
| 1 | 64-bit | fixed64, sfixed64, double |
| 2 | Length-delimited | string, bytes, embedded messages, packed repeated fields |
| 3 | Start group | groups (deprecated) |
| 4 | End group | groups (deprecated) |
| 5 | 32-bit | fixed32, sfixed32, float |

**Varint类型[动态整型]（type为0）**

1.       每个字节第一位表示有无后续字节，有为1，无为0,     (双字节，低字节在前，高字节在后.)

2.       剩余7位倒序合并

举例: 300 的二进制为 10 0101100

第一位：1（有后续） + 0101100

第二位：0（无后续） + 0000010

最终结果： 101011000000010

**Message 结构**

1. 键值型结构（Key-Value）
2. 第一部分为Key值，Varint 结构
3. Key值的后三位表示规则类型的Type值，其他部分和为类型的数字编号
4. 后面紧跟value，value的值依据规则类型不同而不同

举例: required int32 a = 1; 当a值为150时

Key：0000 1000,类型为000，数字编号为0001

Value（Varint类型）：1001 0110  0000 0001

值解码： 000 0001 + 001 0110 = 10010110 = 150

**sint32和sint64类型的编码（ZigZag）**

对于sint32和sint64类型的编码采用ZigZag编码方式，最后一位表示正负情况，即如下：

|  |  |
| --- | --- |
| **原始值** | **编码为** |
| 0 | 0 |
| -1 | 1 |
| 1 | 2 |
| -2 | 3 |
| 2147483647 | 4294967294 |
| -2147483648 | 4294967295 |

解码方式为：

1. 对sint32 -> (n << 1) ^ (n >> 31)
2. 对sint64 -> (n << 1) ^ (n >> 63)

**其他非Varint的数字类型（type为1或5）**

按小端字节序（little-endian）排布（低位字节排放在内存的低地址端，高位字节排放在内存的高地址端）

比如：0x1234ABCD 保存为 0xCD 0xAB 0×34 0×12

**字符串类型（type为2）**

1. 字符串采用UTF-8编码
2. 在声明类型和编号后紧跟一个Varint类型，表示字符串长度
3. 接下来的是字符串内容

比如：required string b = 2; 其中b的值为 testing

结果（16进制）是 12 07 74 65 73 74 69 6e 67

棕色为字符串内容

暗红色为Varint的类型申明及编号

紫色为Varint的长度申明

**内嵌Message类型（type为2）**

内嵌Message类型采用类似字符串的编码方法，只是后面跟的是二进制而不是字符串

比如：

message Test1 {

  required int32 a = 1;

}

message Test3 {

  required Test1 c = 3;

}

其中a.c的值为150

结果为： 1a 03 08 96 01

棕色为Test1的内容

暗红色为Varint的类型申明及编号

紫色为Varint的长度申明

**可重复选项（Repeated）和可选选项（Optional）**

1. 对于可重复项（没有设置[packed=true]），编码的结果里对一个标签编号存在0条或多条key-value结构，并且无需连续和不保证顺序
2. 对于可选项，编码的结果里可能没有该标签编号的key-value结构
3. 对于非可重复项的重复数据的处理方式
4. 对于数字和字符串，只接受最后一次的值，前面的忽略
5. 对于Message，采用合并（Merge）操作，使用后面的值覆盖前面的值

**带有[packed=true]选项的可重复项（type为2）**

可重复项带有[packed=true]后，所有元素打成一个包，使用类似字符串的数据打包形式

message Test4 {

  repeated int32 d = 4 [packed=true];

}

结果如下：

22        // tag (编号 4, 类型 2)

06        // 总长度 (6 bytes)

03        // 第一个元素 (varint 3)

8E 02     // 第二个元素 (varint 270)

9E A7 05  // 第三个元素 (varint 86942)

 到这里就没了，by the way，一些SDK碰到不能识别的数据，将会把它放到最后，比如C++，另一些就直接忽略掉了，比如Python。而且这种设计对协议更新的向后兼容非常的好啊