protobufの中身を覗いてみた

protobufは軽量

protobufについて調べるとよく軽量であることが利点として挙げられている

> think XML, but smaller, faster, and simpler
> Protocol Buffers

→本当？覗いてみよう

バイナリの眺め方

以下のようにしてバイナリを眺めることができる

```
message User {
  int32 id = 1;
}
```

```
import user_pb2

user = user_pb2.User()
user.id = 12
print(" ".join([f'{x:08b}' for x in user.SerializeToString()]))
# 00001000 00000001
```

バイナリを読んでみる

```
00001000 00001100
```

- 1bit目(0) ... フィールド番号の後続があるかどうかのflag
- 2-5bit目(0001) ... フィールド番号 → 1
- 6-8bit目(000) ... Wire type → 0 (VARINT)

Wire type

バイナリを読んでみる

```
00001000 00001100
```

- 9bit目(0) ... フィールドの値に後続があるかどうかのflag
- 10-16bit目(0001100) ... フィールドの値→12であることがわかる

もっと難しいのを読んでみる（protbuf）

```
message User {
  int32 id = 1;
  string name = 2;
  repeated int64 num = 99;
}
```

もっと難しいのを読んでみる（python）

```python
import user_pb2

user = user_pb2.User()
user.name = "A"
user.num.extend([1, 10])
binary = user.SerializeToString()
print(" ".join([f'{x:08b}' for x in binary]))

# 00010010 00000101 01100101 01110101 01100011 01111001 01110100 10011010 00000110 00000010 00000001 00001010
```

もっと難しいのを読んでみる（前半）

```
00010010 00000101 01100101 01110101 01100011 01111001 01110100
```

- 2-5bit目(0010) ... フィールド番号 → 2
- 6-8bit目(010) ... Wire type → 2(LEN)
- 9-16bit目(0000101) ... フィールドの値の長さ → 5bytes
- 残り ... 0x65 0x75 0x63 0x79 0x74 → eucyt

もっと難しいのを読んでみる（後半）

```
10011010 00000110 00000010 00000001 00001010
```

- 1bit目(1) ... 後続byteあり
- 2-4, 9-16bit目(0000110 0011) ... フィールド番号 → 99
- 5-8bit目(010) ... Wire type → 2(LEN)
- 9-16bit目(0000010) ... フィールドの値の長さ → 2bytes
- 18-24bit目(0000001) ... フィールドの値 → 1
- 26-32bit目(0001010) ... フィールドの値 → 10

最後にTips

小さな負数を扱う場合は、sint32, sint64を使うと良い

int32で-1は `ffff ffff ffff ffff ff01`

一方sint32はzigzag encodingなので、-1は `1` になる

参考文献

Protobuf Encording
つくって学ぶ Protocol Buffers エンコーディング
protobufのバイナリを眺めてみる