Protobuf序列化原理

一、 Protobuf序列化原理简介

1.1序列化

序列化是将数据结构或对象转换成二进制字节流的过程。
Protobuf对于不同的字段类型采用不同的编码方式和数据存储方式对消息字段进行序列化，以确保得到高效紧凑的数据压缩。
Protobuf序列化过程如下：
(1)判断每个字段是否有设置值，有值才进行编码。
(2)根据字段标识号与数据类型将字段值通过不同的编码方式进行编码。
(3)将编码后的数据块按照字段类型采用不同的数据存储方式封装成二进制数据流。

1.2反序列化

反序列化是将在序列化过程中所生成的二进制字节流转换成数据结构或者对象的过程。
Protobuf反序列化过程如下：
(1)调用消息类的parseFrom(input)解析从输入流读入的二进制字节数据流。
(2)将解析出来的数据按照指定的格式读取到C++、Java、Phyton对应的结构类型中。

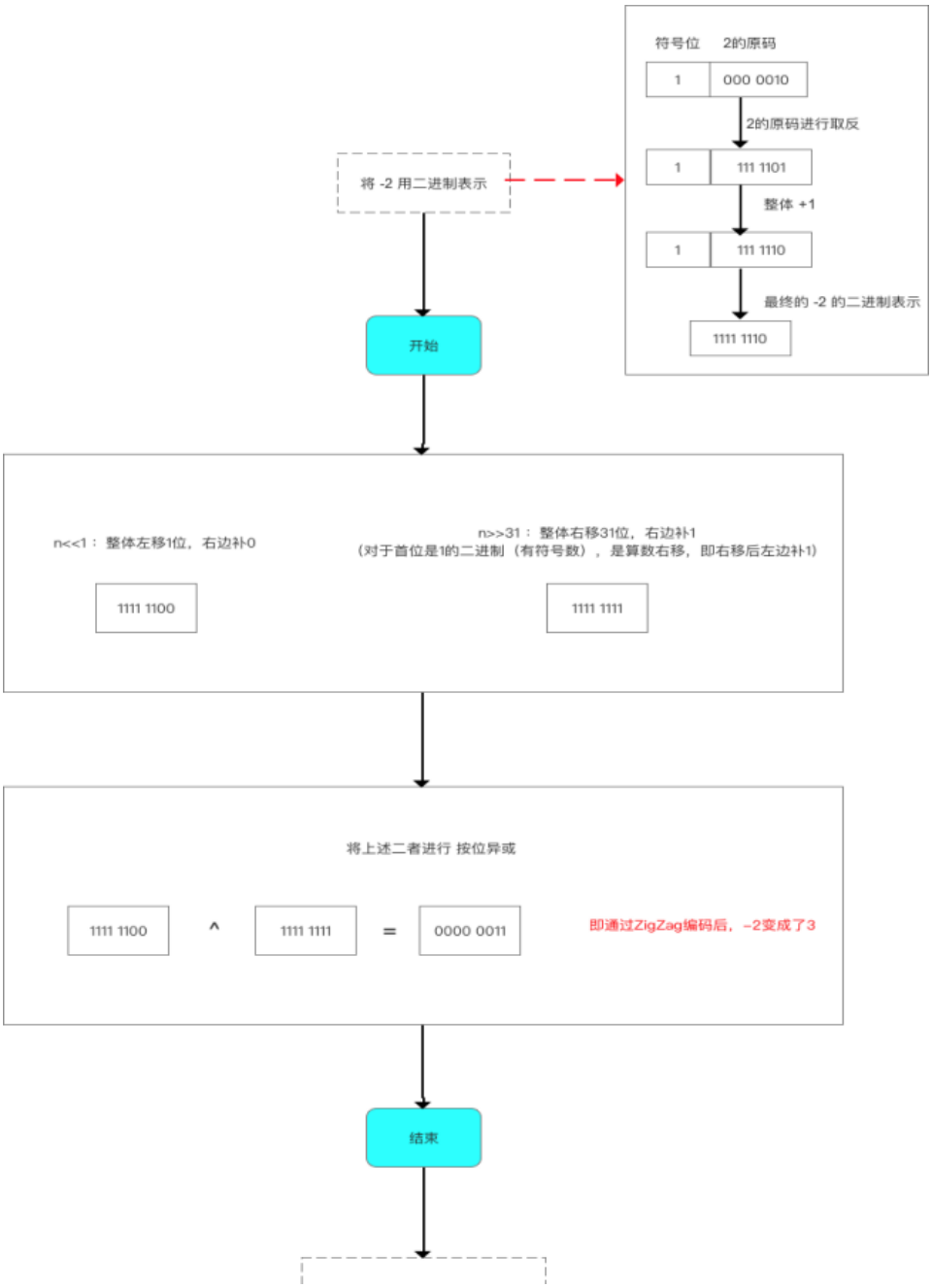
二、 Protobuf编码方式

2.1Varint编码

Varint编码是一种变长的编码方式，编码原理是用字节表示数字，值越小的数字，使用越少的字节数表示。因此，可以通过减少表示数字的字节数进行数据压缩。
对int32类型的数字，一般需要4个字节表示。如果采用Varint编码，对于很小的int32类型数字，则可以用1个字节来表示；虽然大的数字会需要5个字节来表示，但大多数情况下，消息都不会有很大的数字，所以采用Varint编码方式总是可以用更少的字节数来表示数字。
Varint编码后每个字节的最高位都有特殊含义：
A、如果是1，表示后续的字节也是数字的一部分。
B、如果是0，表示本字节是最后一个字节，且剩余7位都用来表示数字。
当使用Varint解码时时，只要读取到最高位为0的字节时，表示本字节是一个值经Varint编码后得到的字节流的最后一个字节。
在计算机内，负数一般会被表示为很大的整数，因为计算机定义负数的符号位为数字的最高位，如果采用Varint编码方式表示一个负数，那么一定需要5个byte(因为负数的最高位是1，会被当做很大的整数处理)
Protobuf定义了sint32 / sint64类型表示负数，通过先采用Zigzag编码(将有符号数转换成无符号数)，再采用Varint编码，从而用于减少编码后的字节数。
对于一个int32类型的值300的Varint编码如下：
300的二进制编码为：100101100(256+32+8+4)
从字节流末尾取出7bit并在最高位增加1构成一个字节：[1]010 1100
从字节流末尾取出7bit并在最高位增加1构成一个字节，如果是最后一个字节增加0：[0]0000010
两字节为：[0]0000010 [1]010 1100
转换为小端模式：10101100 00000010
编码结果：1010 1100 0000 0010

2.2Zigzag编码

Zigazg编码是一种变长的编码方式，其编码原理是使用无符号数来表示有符号数字，使得绝对值小的数字都可以采用较少字节来表示，特别对表示负数的数据能更好地进行数据压缩。
Zigzag编码对Varint编码在表示负数时不足的补充，从而更好的帮助Protobuf进行数据的压缩。因此，如果提前预知字段值是可能取负数的时候，需要采用sint32/sint64数据类型。
Protobuf通过Varint和Zigzag编码后，大大减少了字段值占用字节数。
-2的Zigzag过程如下：



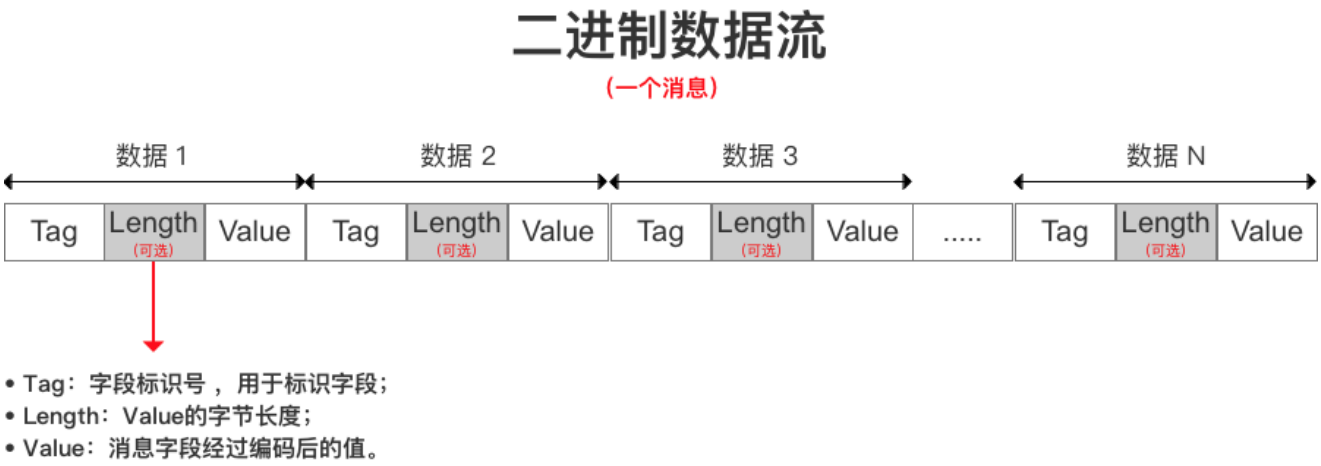
gRPC快速入门(二)——Protobuf序列化原理解析

三、 Protobuf数据存储方式

3.1T-L-V数据存储方式

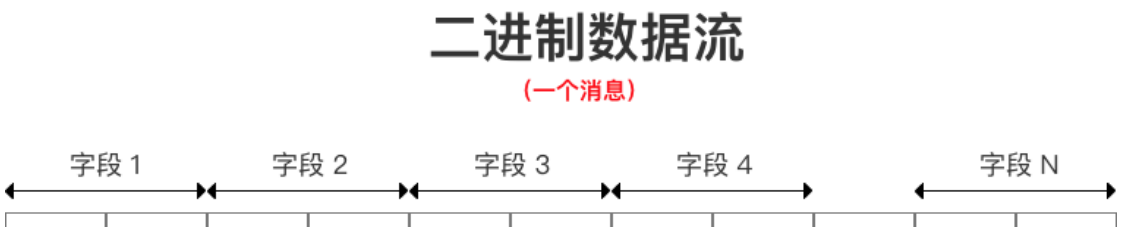
T-L-V(Tag - Length - Value)，即标识符-长度-字段值的存储方式，其原理是以标识符-长度-字段值表示单个数据，最终将所有数据拼接成一个字节流，从而实现数据存储的功能。
其中Length可选存储，如储存Varint编码数据就不需要存储Length，此时为T-V存储方式。

T-L-V 存储方式的优点：
A、不需要分隔符就能分隔开字段，减少了分隔符的使用。
B、各字段存储得非常紧凑，存储空间利用率非常高。
C、如果某个字段没有被设置字段值，那么该字段在序列化时的数据中是完全不存在的，即不需要编码，相应字段在解码时才会被设置为默认值。



3.2T-V数据存储方式

消息字段的标识号、数据类型、字段值经过Protobuf采用Varint和Zigzag编码后，以T-V(Tag-Value)方式进行数据存储。
对于Varint与Zigzag编码方式编码的数据，省略了T-L-V中的字节长度Length。



Tag是消息字段标识符和数据类型经Varint与Zigzag编码后的值，因此Tag存储了字段的标识符(field_number)和数据类型(wire_type)，即Tag = 字段数据类型(wire_type) + 标识号(field_number)。
Tag占用一个字节的长度(如果标识符大于15，则占用多一个字节的位置)，字段数据类型(wire_type)占用3个bit，字段标识符(field_number)占用4个bit，最高位用于Varint编码保留。

```
Tag = (field_number << 3) | wire_type
enum WireType {
    WIRETYPE_VARINT = 0,
    WIRETYPE_FIXED64 = 1,
    WIRETYPE_LENGTH_DELIMITED = 2,
    WIRETYPE_START_GROUP = 3,
    WIRETYPE_END_GROUP = 4,
    WIRETYPE_FIXED32 = 5
  };
```

解码时，Protobuf根据Tag将Value对应于消息中的字段。

```
message person
{
  required int32    id = 1;
  // wire type = 0，field_number =1
  required string    name = 2;
  // wire type = 2，field_number =2
 }
```

对于Person消息的name字段的Tag编码如下：

```
nameTag = 2 << 3 | 2
nameTag = 0001 0010
```

根据Tag解码得到filed_number、wire_type：

```
nameTag = 0001 0010
field_number = nameTag >> 3
field_number = 0010
wire_type = nameTag & 3
wire_type = 010
```

四、Protobuf序列化原理解析

Protobuf对于数据存储的三大原则：
(1)Protocol Buffer将消息中的每个字段进行编码后，利用T - L - V 存储方式进行数据的存储，最终得到一个二进制字节流。
(2)ProtoBuf对于不同数据类型采用不同的序列化方式(数据编码方式与数据存储方式)
Protobuf对于不同的字段类型采用不同的编码和数据存储方式对消息字段进行序列化，以确保得到高效紧凑的数据压缩。不同类型的数据采用的编码方式和存储方式如下：

Wire Type 值	编码方式	编码长度	存储方式	代表的数据类型
0	Varint （负数时以Zigzag辅助编码）	变长（1–10个字节）	T – V	• int32, int64, uint32, uint64,bool, enum • sint32, sint64(负数时使用)
1	64–bit	固定8个字节		fixed64, sfixed64, double
2	Length–delimi	变长	T – L – V	string, bytes, embedded messages, packed repeated fields
3	Start group	已弃用	已弃用	Groups (已弃用)
4	End group			

wire_type
只有六种类型，所以用三位二进制数完全足够表示。
Tag = (field_number << 3) | wire_type

对于Varint编码数据的存储，不需要存储字节长度Length，使用T-V存储方式进行存储；对于采用其它编码方式(如LENGTH_DELIMITED)的数据，使用T-L-V存储方式进行存储。
(3)ProtoBuf对于数据字段值的独特编码方式与T-L-V数据存储方式，使得 ProtoBuf序列化后数据量体积极小。

2、WireType=0的序列化
WireType=0的类型包括int32，int64，uint32，unint64，bool，enum以及sint32和sint64。
编码方式采用Varint编码(如果为负数，采用Zigzag辅助编码)，数据存储方式使用T-V方式存储二进制字节流。

3、WireType=1的序列化
WireType=1的类型包括fixed64，sfixed64，double。
编码方式采用64bit编码(编码后数据大小为64bit，高位在后，低位在前)，数据存储方式使用T-V方式存储二进制字节流。

4、WireType=2的序列化
WireType=2的类型包括string，bytes，嵌套消息，packed repeated字段。
对于编码方式，标识符Tag采用Varint编码，字节长度Length采用Varint编码，string类型字段值采用UTF-8编码，嵌套消息类型的字段值根据嵌套消息内部的字段数据类型进行选择，
数据存储方式使用T-L-V方式存储二进制字节流。

5、WireType=5的序列化
WireType=5的类型包括fixed32，sfixed32，float。
编码方式采用32bit编码(编码后数据大小为32bit，高位在后，低位在前)，数据存储方式使用T-V方式存储二进制字节流。

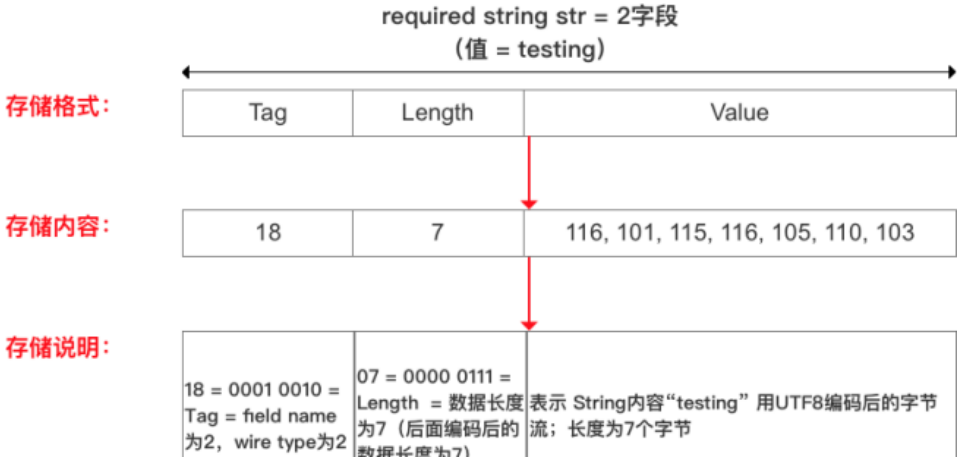
五Protobuf序列化示例

5.1String类型

String类型字段的值使用UTF-8编码。消息数据流如下：

gRPC快速入门(二)——Protobuf序列化原理解析

```
message Test
{
   required string str = 2;
}

// 将str设置为：testing
Test.setStr("testing")

// 经过protobuf编码序列化后的数据以二进制的方式输出
// 输出为：18, 7, 116, 101, 115, 116, 105, 110, 103
```



gRPC快速入门(二)——Protobuf序列化原理解析

5.2嵌套消息类型

嵌套消息类型采用T-L-V的存储方式，外部消息的V即为嵌套消息的字段，在T-L-V的V中嵌套了一系列的T-L-V。
编码方式：字段值(即V)根据字段的数据类型采用不同编码方式。

gRPC快速入门(二)——Protobuf序列化原理解析

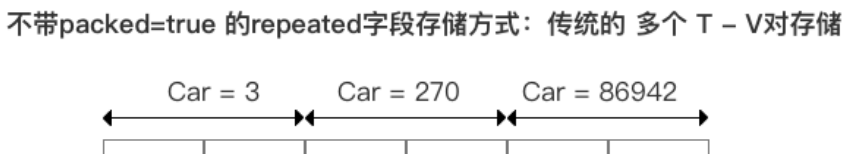
```
message Test2
{
   required string str = 1;
   required int32 id1 = 2;
}

message Test3 {
  required Test2 c = 1;
}

// 将Test2中的字段str设置为：testing
// 将Test2中的字段id1设置为：296
// 编码后的字节为：10 ， 12 ， 18，7，116, 101, 115, 116, 105, 110, 103，16，-88，2
```

gRPC快速入门(二)——Protobuf序列化原理解析

5.3通过packed修饰的 repeat 字段

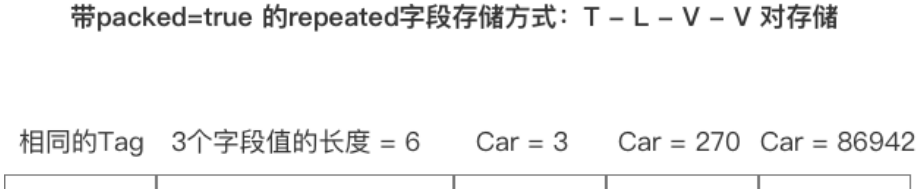
```
message Test
{
   repeated int32 Car = 4 ;
   // 表达方式1：不带packed=true
   repeated int32 Car = 4 [packed=true];
   // 表达方式2：带packed=true
}

Test.setCar(3);
Test.setCar(270);
Test.setCar(86942);
```

如果序列化时对多个 T - V对存储(不带packed=true)，则会导致Tag的冗余，即相同的Tag存储多次。



gRPC快速入门(二)——Protobuf序列化原理解析

为了解决Tag数据冗余，采用带packed=true的repeated字段存储方式，即将相同的Tag只存储一次、添加repeated字段下所有字段值的长度Length、连续存储repeated字段值，组成一个大的Tag - Length - Value -Value -Value对，即T - L - V - V - V对。



gRPC快速入门(二)——Protobuf序列化原理解析

通过采用带packed=true 的 repeated字段存储方式，从而更好地压缩序列化后的数据长度。

六Protobuf使用建议

基于Protobuf序列化原理分析，为了有效降低序列化后数据量的大小，可以采用以下措施：

(1)多用 optional或 repeated修饰符

若optional 或 repeated 字段没有被设置字段值，那么该字段在序列化时的数据中是完全不存在的，即不需要进行编码，但相应的字段在解码时会被设置为默认值。

(2)字段标识号(Field_Number)尽量只使用1-15，且不要跳动使用

Tag是需要占字节空间的。如果Field_Number>16时，Field_Number的编码就会占用2个字节，那么Tag在编码时就会占用更多的字节；如果将字段标识号定义为连续递增的数值，将获得更好的编码和解码性能。

(3)若需要使用的字段值出现负数，请使用sint32/sint64，不要使用int32/int64。

采用sint32/sint64数据类型表示负数时，会先采用Zigzag编码再采用Varint编码，从而更加有效压缩数据。

(4)对于repeated字段，尽量增加packed=true修饰

增加packed=true修饰，repeated字段会采用连续数据存储方式，即T - L - V - V -V方式。

转载文章