通信协议之序列化

分类： 网络与安全　2012-07-07 15:15:34

stevenrao——2012-07-07于深圳

通信协议可以理解两个节点之间为了协同工作实现信息交换，协商一定的规则和约定，例如规定字节序，各个字段类型，使用什么压缩算法或加密算法等。常见的有tcp，udo，http，sip等常见协议。协议有流程规范和编码规范。流程如呼叫流程等信令流程，编码规范规定所有信令和数据如何打包/解包。

编码规范就是我们通常所说的编解码，序列化。不光是用在通信工作上，在存储工作上我们也经常用到。如我们经常想把内存中对象存放到磁盘上，就需要对对象进行数据序列化工作。

本文采用先循序渐进，先举一个例子，然后不断提出问题-解决完善，这样一个迭代进化的方式，介绍一个协议逐步进化和完善，最后总结。看完之后，大家以后在工作就很容易制定和选择自己的编码协议。

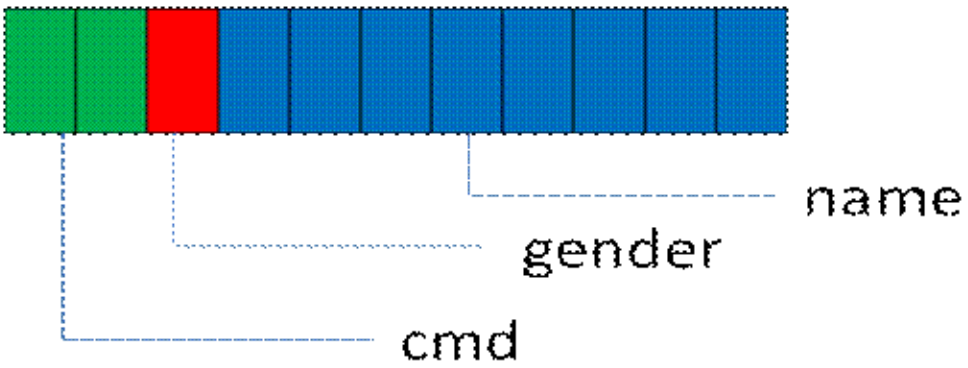
一、**紧凑模式**

本文例子是A和B通信，获取或设置基本资料，一般开发人员第一步就是定义一个协议结构:

```
struct userbase
{
unsigned short cmd;//1-get, 2-set, 定义一个short，为了扩展更多命令(理想那么丰满)
unsigned char gender; //1 – man , 2-woman, 3 - ??
char name[8]; //当然这里可以定义为 string name；或len + value 组合，为了叙述方便，就使用简单定长数据
}
```

在这种方式下，A基本不用编码，直接从内存copy出来，再把cmd做一下网络字节序变换，发送给B。B也能解析，一切都很和谐愉快。

这时候编码结果可以用图表示为(1格一个字节)



这种编码方式，我称之为**紧凑模式**，意思是除了数据本身外，没有一点额外冗余信息，可以看成是Raw Data。在dos年代，这种使用方式非常普遍，那时候可是内存和网络都是按K计算，cpu还没有到1G。如果添加额外信息，不光耗费捉襟见肘的cpu，连内存和带宽都伤不起。

二、**可扩展性**

有一天，A在基本资料里面加一个生日字段，然后告诉B

struct userbase

{

unsigned short cmd;

unsigned char gender;

unsigned int birthday;

char name[8];

}

这是B就犯愁了，收到A的数据包，不知道第3个字段到底是旧协议中的name字段，还是新协议中birthday。这是后A，和B终于从教训中认识到一个协议重要特性——**兼容性和可扩展性**。

于是乎，A和B决定废掉旧的协议，从新开始，制定一个以后每个版本兼容的协议。方法很简单，就是加一个version字段。

struct userbase

{

unsigned short version;

unsigned short cmd;

unsigned char gender;

unsigned int birthday;

char name[8];

}

这样，A和B就松一口气，以后就可以很方便的扩展。增加字段也很方便。这种方法即使在现在，应该还有不少人使用。

二、**更好的可扩展性**

过了一段较长时间，A和B发现又有新的问题，就是没增加一个字段就改变一下版本号，这还不是重点，重点是这样代码维护起来相当麻烦，每个版本一个case分支，到了最好，代码里面case 几十个分支，看起来丑陋而且维护起来成本高。

A 和 B仔细思考了一下，觉得光靠一个version维护整个协议，不够细，于是觉得为每个字段**增加一个额外信息——tag,虽然增加内存和带宽，但是现在已经不像当年那样，可以容许这些冗余，换取易用性**。

struct userbase

{

1 unsigned short version;

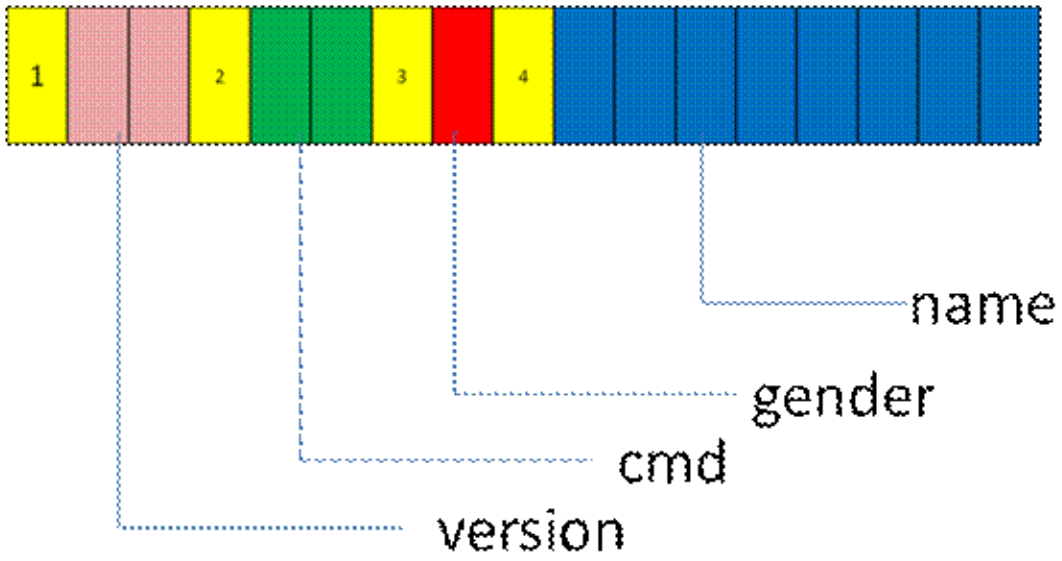
2 unsigned short cmd;

3 unsigned char gender;

4 unsigned int birthday;

5 char name[8];

}

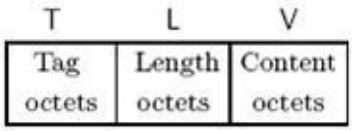


制定完这些协议后，A和B很得意，觉得这个协议不错，可以自由的增加和减少字段。随便扩展。

现实总是很残酷的，不久就有新的需求，name使用8个字节不够，最大长度可能会达到100个字节，A和B就愁怀了，总不能即使叫“steven”的人，每次都按照100个字节打包，虽然不差钱，也不能这样浪费。

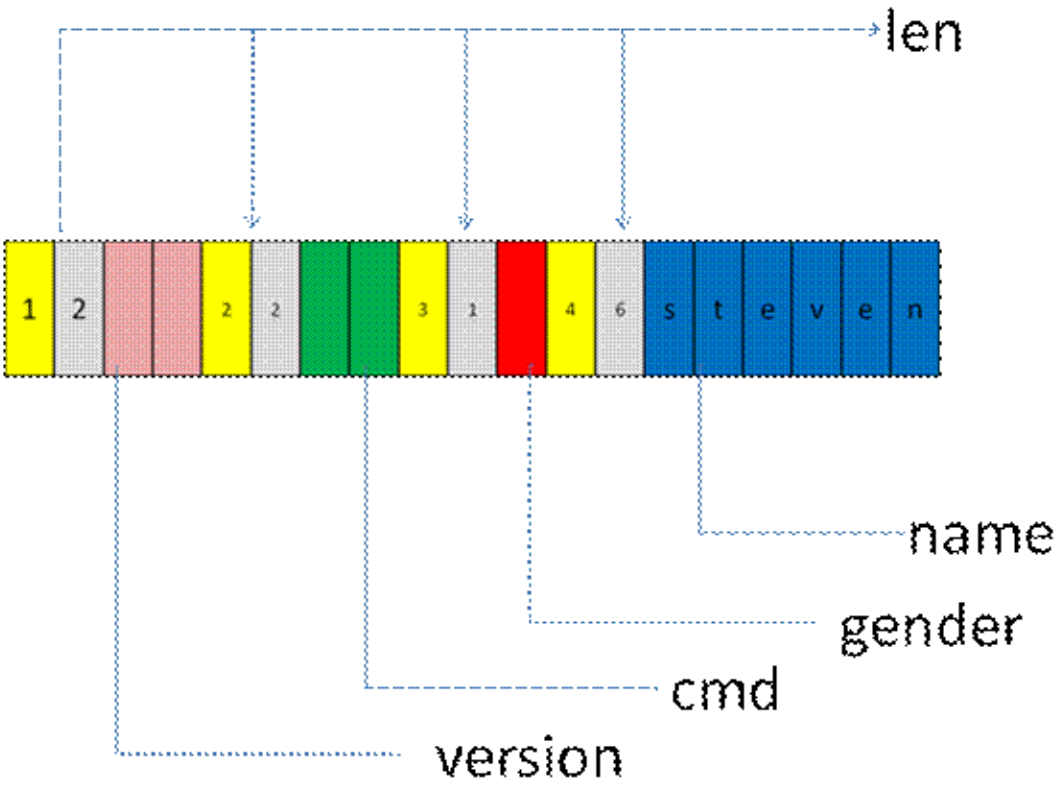
于是A和B寻找各方资料，找到了ANS.1编码规范，好东西啊.. ASN.1是一种ISO/ITU-T 标准。其中一种编码BER（Basic Encoding Rules）简单好用，它使用三元组编码，简称TLV编码。

每个字段编码后内存组织如下



字段可以是结构，即可以嵌套

A和B使用TLV打包协议后，数据内存组织大概如下:



TLV具备了很好可扩展性，很简单易学。同时也具备了缺点，因为其增加了2个额外的冗余信息，tag和len，特别是如果协议大部分是基本数据类型int ,short, byte. 会浪费几倍存储空间。另外Value具体是什么含义，需要通信双方事先得到描述文档，即TLV不具备结构化和自解释特性。

三、**自解释性**

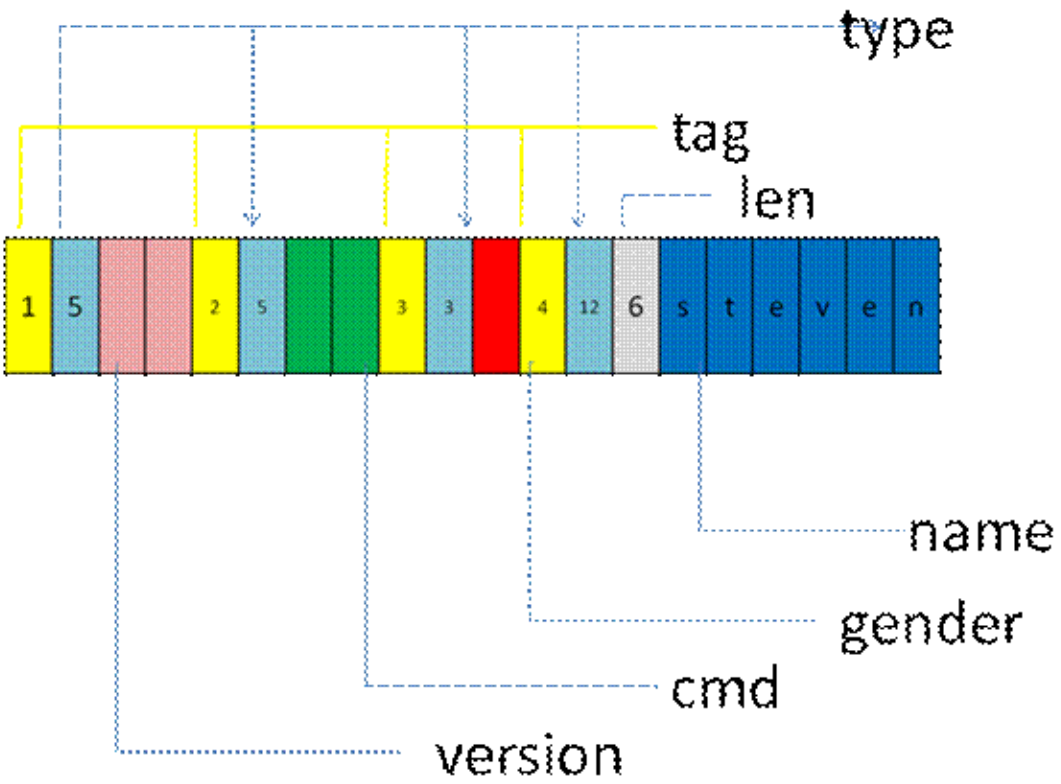
当A和B采用TLV协议后，似乎问题都解决了。但是还是觉得不是很完美，决定增加自解释特性，这样抓包就能知道各个字段类型，不用看协议描述文档。这种改进的类型就是 TT[L]V（tag，type，length，value），其中L在type是定长的基本数据类型如int,short, long, byte时候，因为其长度是已知的，所以L不需要。

于是定义了一些type值如下

类型	Type值	类型描述
bool	1	布尔值
int8	2	带符号的一个字符
uint8	3	带符号的一个字符
int16	4	16位有符号整型
uint16	5	16位无符号整型
int32	6	32位有符号整型
uint32	7	32位无符号整型

…		
string	12	字符串或二进制序列
struct	13	自定义的结构，嵌套使用
list	14	有序列表
map	15	无序列表

按照ttlv序列化后，内存组织如下



改完后，A和B发现，的确带来很多好处，不光可以随心所以的增删字段，还可以修改数据类型，例如把cmd改成int cmd；可以无缝兼容。真是太给力了。

三、**跨语言特性**

有一天来了一个新的同事C，他写一个新的服务，需要和A通信，但是C是用java或PHP的语言，没有无符号类型，导致负数解析失败。为了解决这个问题，A重新规划一下协议类型，做了有些剥离语言特性，定义一些共性。对使用类型做了强制性约束。**虽然带来了约束，但是带来通用型和简洁性，和跨语言性**，大家表示都很赞同，于是有了一个类型(type)规范。

类型	Type值	类型描述
bool	1	布尔值
int8	2	带符号的一个字符
int16	3	16位有符号整型

int32	4	32位有符号整型
…		
string	12	字符串或二进制序列
struct	13	自定义的结构，嵌套使用
list	14	有序列表
map	15	无序列表

四、代码自动化 ——IDL语言的产生

但是A和B发现了新的烦恼，就是每搞一套新的协议，都要从头编解码，调试，虽然TLV很简单，但是写编解码是一个毫无技术含量的枯燥体力活，一个非常明显的问题是，由于大量copy/past,不管是对新手还是老手，非常容易犯错，一犯错，定位排错非常耗时。于是A想到使用工具自动生成代码。

IDL（Interface Description Language），它是一种描述语言，也是一个中间语言，IDL一个使命就是规范和约束，就像前面提到，规范使用类型，提供跨语言特性。通过工具分析idl文件，生成各种语言代码

Gencpp.exe sample.idl 输出 sample.cpp sample.h

Genphp.exe sample.idl 输出 sample.php

Genjava.exe sample.idl 输出 sample.java

是不是简单高效J

四、总结

大家看到这里，是不是觉得很面熟。是的，协议讲到最后，其实就是和facebook的thrift和google protocol buffer协议大同小异了。包括公司无线使用的jce协议。咋一看这些协议的idl文件，发现几乎是一样的。只是有些细小差异化。

这些协议在一些细节上增加了一些特性：

1、压缩，这里压缩不是指gzip之类通用压缩，是指针对整数压缩，如int类型，很多情况下值是小于127（值为0的情况特别多），就不需要占用4个字节，所以这些协议做了一些细化处理，把int类型按照情况，只使用1/2/3/4字节，实际上还是一种ttlv协议。

2、reuire/option 特性: 这个特性有两个作用，1、还是压缩，有时候一个协议很多字段，有些字段可以带上也可以不带上，不赋值的时候不是也要带一个缺省值打包，这样很浪费，如果字段是option特性，没有赋值的话，就不用打包。2、有点逻辑上约束功能，规定哪些字段必须有，加强校验。

序列化是通信协议的基础，不管是信令通道还是数据通道，还是rpc，都需要使用到。在设计协议早期就考虑到扩展性和跨语言特性。会为以后省去不少麻烦。

Ps

本篇主要介绍二进制通信协议序列化，没有讲文本协议。从某种意义来讲，文本协议天生具有兼容和可扩展性。不像二进制需要考虑那么多问题。文本协议易于调试（如抓包就是可见字符，telnet即可

调试，数据包可以手工生成不借助特殊工具），简单易学是其最强大的优势。

二进制协议优势就是性能和安全性。但是调试麻烦。

两者各有千秋，按需选择。(stevenrao)

上一篇：google 搜索
下一篇：漫谈linux文件IO