数据编码演化

本文内容参考ddia(designing data-intensive applications)第四章

一般来讲，在程序中数据至少有以下两种形式：

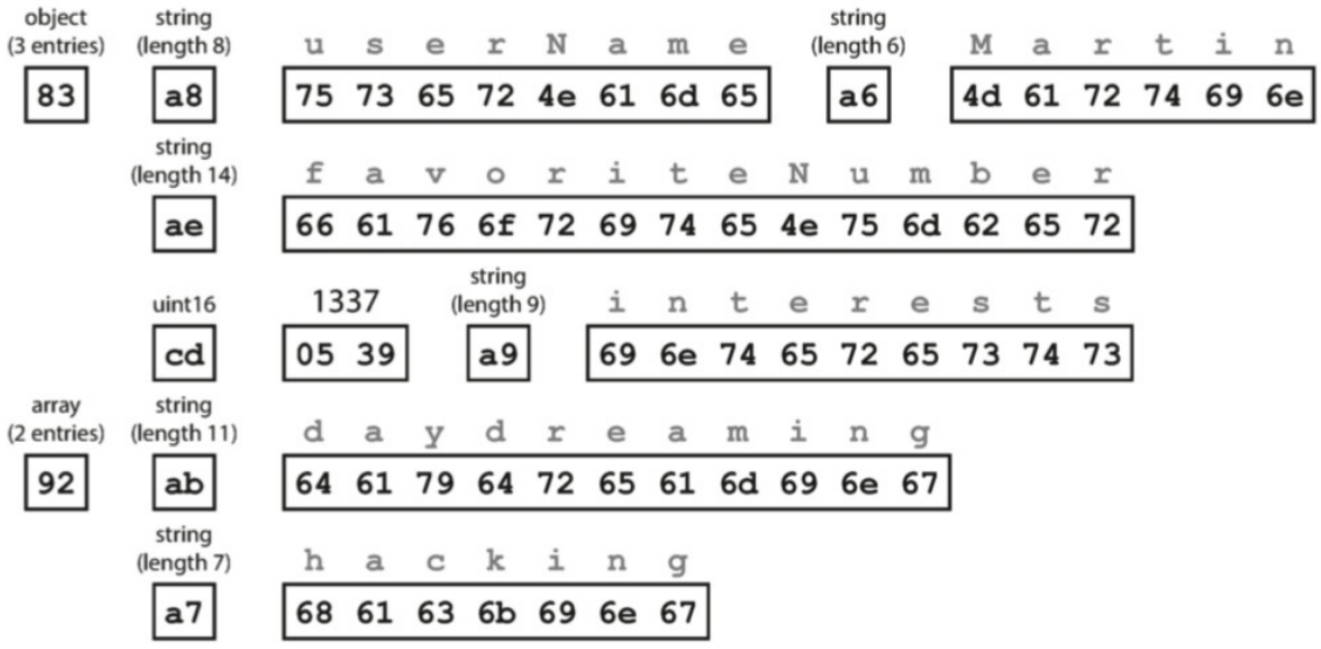
- 内存中以各种数据结构的形式存在
- 进行io操作时，必须将数据进行编码

JSON是常用的编码方式，下面是一个JSON记录

```
{
    "userName": "Martin",
    "favoriteNumber": 1337,
    "interests": ["daydreaming", "hacking"]
}
```

接下来，我们将以这个例子，说明各种数据编码形式的方式，这个文本去掉空格占用的空间为81字节

1. MessagePack



MessagePack是一种JSON的二进制编码，和文本形式其实区别不大，以“userName"为例，先用0xa8表示接下来是一个长度为8的string，然后跟着8字节的ASCII表示字段名。

在这个例子中，MessagePack编码后长度为66字节，和文本JSON编码相比，缩小的空间有限，其他的JSON二进制编码方式也都类似。

2. Thrift

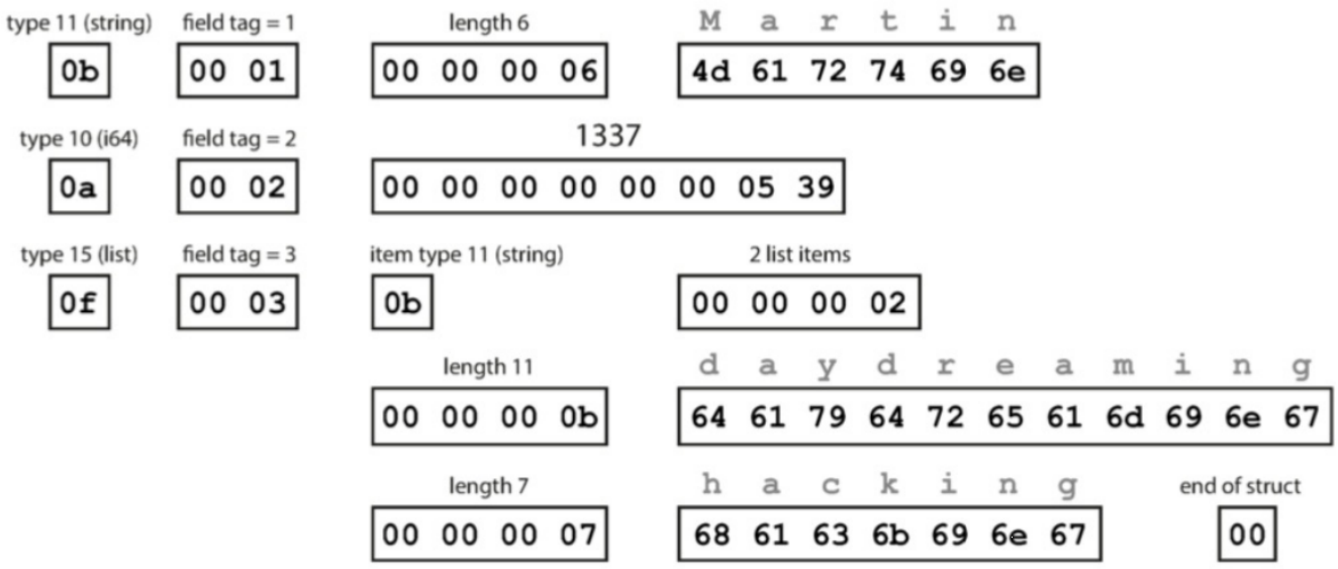
从上面的例子我们可以看出JSON编码的效率并不高，那么是不是有其他更高效的方式呢，下面介绍我们常用的Thrift是怎么对上述例子进行编码的，Thrift有两种二进制编码格式，分别为BinaryProtocol和CompactProtocol。

首先我们上述例子的idl为：

```
struct Person {
    1: required string userName
    2: optional i64 favoriteNumber
    3: optional list interests
}
```

2.1 BinaryProtocol

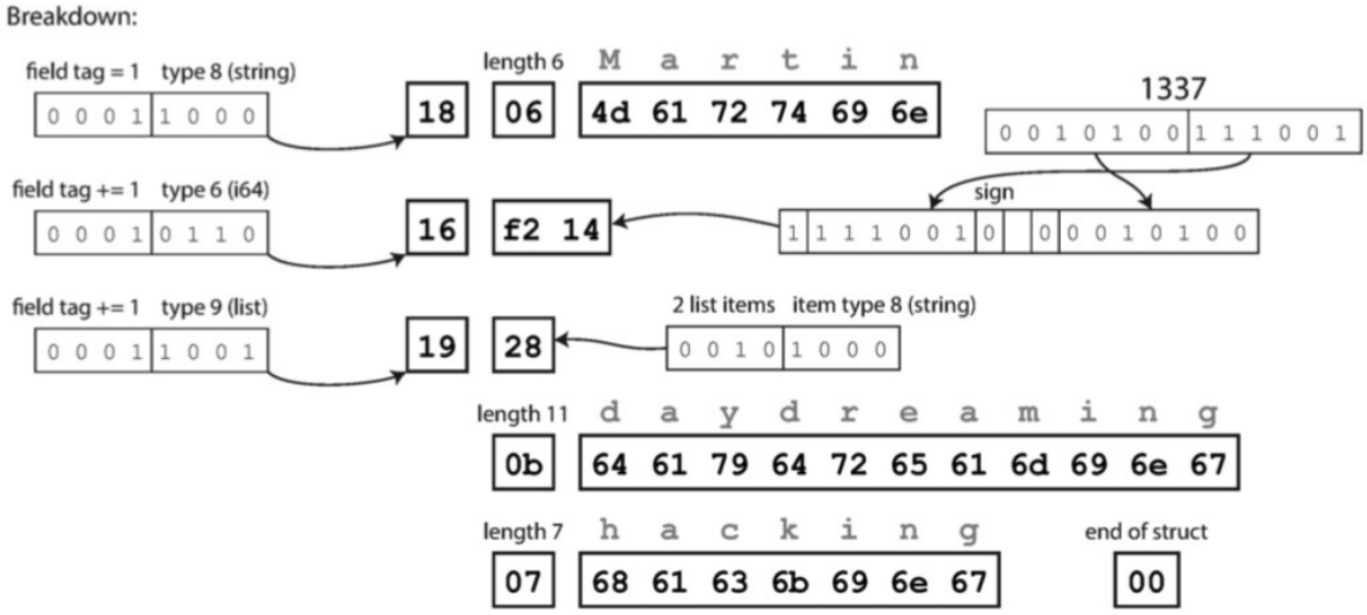
下面先介绍一下BinaryProtocol，同样对于上述例子，BinaryProtocol是这样编码的：



从上图可以看出，BinaryProtocol不再存放字段名称了，而是用tag进行标识(这里我理解的是服务端和客户端都是自己产生的，所以没有必要有名称了)，其他部分差别不大，这部分长度为59B

2.2 CompactProtocol

接下来介绍CompactProtocol，编码形式如下：



这部分做了两件事来进一步对编码进行压缩，首先将filed tag和type压缩到一个字节实现了，同时避免tag超出4bit范围，将第一个之后到表示为前一个到偏移。此外，对于1337这个数字，也不再采用8字节了，而是采用了变长编码，每个字节到第一位用来表示是否还有更多到字节，这样1337最后只用两个字节就可以表示完全。这种编码方式最后长度为34字节

此外，字段是required还是optional不会影响编码形式，但required字段如果未填充，则会失败。

2.3 thrift与兼容性

从编码方式我们可以看出：

- 可以改变字段的名称，因为编码使用tag进行标示的
- 相应的，不能随意更改tag
- 向前兼容：添加新的字段的话，如果旧代码读取新代码，遇到了不识别的标记号，它会简单的忽略该字段
- 向后兼容：添加字段时，新代码可以读取旧代码的数据，但是，如果新添加的字段是required字段，那新代码读取旧代码的数据检查会失败。
- 删除字段类似，只能删除可选字段，且删除后不要再使用该tag，防止旧代码还会生成有该tag的数据