**Hessian 2.0序列化协议规范**

翻译: Edison peng

**目录**

[1.概述 4](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068030)

[2.设计目标 4](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068031)

[3. Hessian语法 4](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068032)

[4. 序列化 6](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068033)

[4.1. 二进制数据 7](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068034)

[4.1.1. 压缩格式：短二进制 7](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068035)

[4.1.2. Binary实例 7](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068036)

[4.2. boolean 7](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068037)

[4.3. date 8](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068038)

[4.3.1. Date实例 8](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068039)

[4.4. double 8](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068040)

[4.4.1. 压缩格式：double表示的0 8](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068041)

[4.4.2. 压缩格式：double 表示的1 8](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068042)

[4.4.3. 压缩格式：单字节double 9](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068043)

[4.4.4. 压缩格式：short型double 9](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068044)

[4.4.5. float型double 9](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068045)

[4.4.6. Double实例 9](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068046)

[4.5. int 9](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068047)

[4.5.1. 单字节整型 10](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068048)

[4.5.2. 双字节整型 10](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068049)

[4.5.3. 三字节整型 10](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068050)

[4.5.4. 整型实例 10](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068051)

[4.6. list 11](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068052)

[4.6.1. 压缩格式: repeated list 11](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068053)

[4.6.2. List实例 11](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068054)

[4.7. long 12](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068055)

[4.7.1. 压缩格式: 单字节long 12](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068056)

[4.7.2. 压缩格式: 双字节long 12](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068057)

[4.7.3. 压缩格式: 3字节long 12](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068058)

[4.7.4. 压缩格式: 四字节long 13](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068059)

[4.7.5. long实例 13](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068060)

[4.8. map 13](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068061)

[4.8.1. Map实例 13](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068062)

[4.9. null 14](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068063)

[4.10. 对象(object) 15](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068064)

[4.10.1. 压缩格式: class定义 15](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068065)

[4.10.2. 压缩格式: 对象实例 15](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068066)

[4.10.3. 对象实例 15](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068067)

[4.11. 引用(ref) 16](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068068)

[4.11.1. 压缩格式: 双字节引用 17](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068069)

[4.11.2. 压缩格式: 三字节引用 17](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068070)

[4.11.3. 引用实例 17](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068071)

[4.12. string 18](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068072)

[4.12.1. 压缩格式: 短字符串 18](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068073)

[4.12.2 字符串实例 18](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068074)

[4.13. 类型(type) 18](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068075)

[4.14. 压缩格式: 类型引用 18](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068076)

[5. 引用表(Reference Map) 19](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068077)

[5.1. 值引用 19](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068078)

[5.2. class引用 19](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068079)

[5.3. type引用 19](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068080)

[6. 字节码映射表(Bytecode map) 19](http://writeblog.csdn.net/Editor/FCKeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_richTextEditor_richTextEditor&Toolbar=Default#_Toc178068081)

**1.概述**

Hessian是一个轻量级的,自定义描述的二进制RPC协议。Hessian主要用作面向对象的消息通信。

**2.设计目标**

Hessian的初衷是支持动态类型，格式紧凑，跨语言.

Hessian协议的设计目标如下:

  可序列化类型必须是可以自描述的, i.e. 不需要额外的模式(schema)或接口定义.

  必须是语言独立的, 包括对脚本语言的支持.

  It must be readable or writable in a single pass.

  必须设计紧凑.

  必须足够简单，以便测试和实现.

  高效率.

  必须支持Unicode字符集.

  支持8-bit二进制数据，without escaping or using attachments.

  必须支持加密, 压缩, 签名 and transaction context envelopes.

**3. Hessian语法**

序列化语法

 #starting production

top ::=value

 #分割成64k每chunk的8-bit二进制数据

binary ::= 'b' b1 b0 <binary-data> binary #不是最后一个chunk

 ::= 'B' b1 b0 <binary-data> #最后一个chunk

 ::= [x20-x2f] <binary-data> #长度范围为 0-15

 #boolean true/false

boolean ::= 'T'

 ::= 'F'

 #对象的定义(compact map)

class-def ::= 'O' type int string\*

 #time in UTC encoded as 64-bit long milliseconds since epoch

date ::= 'd' b7 b6 b5 b4 b3 b2 b1 b0

 #64-bit IEEE double

double ::= 'D' b7 b6 b5 b4 b3 b2 b1 b0

 ::= x67 #0.0

 ::= x68 #1.0

 ::= x69 b0 #byte表示的double(-128.0 to 127.0)

 ::= x6a b1 b0 #short表示的double

 ::= x6b b3 b2 b1 b0 #32-bit float表示的double

 #32-bit 有符号整型

int ::= 'I' b3 b2 b1 b0

 ::= [x80-xbf] #-x10 to x3f

 ::= [xc0-xcf] b0 #-x800 to x7ff

 ::= [xd0-xd7] b1 b0 #-x40000 to x3ffff

 # list/vector length

length ::= 'l' b3 b2 b1 b0

 ::= x6e int

 # list/vector

list ::= 'V' type? length? value\* 'z'

 ::= 'v' int int value\* #第一个int表示类型引用, 第二个int表示长度

 #64-bit有符号long

long ::= 'L' b7 b6 b5 b4 b3 b2 b1 0

 ::= [xd8-xef] #-x08 to x0f

 ::= [xf0-xff] b0 #-x800 to x7ff

 ::= [x38-x3f] b1 b0 #-x40000 to x3ffff

 ::= x77 b3 b2 b1 b0 #32-bit 整型表示的long

 #map/object

map ::= 'M' type? (value value)\* 'z' #key, value map pairs

 # null value

null ::= 'N'

 #对象实例

object ::= 'o' int value\*

 #值引用

ref ::= 'R' b3 b2 b1 b0 # 对流中第n个map/list/object的引用

 ::= x4a b0 # 对map/list/object的引用，范围为1-255th

 ::= x4b b1 b0 # 对map/list/object 的引用，范围为1-65535th

 #UTF-8 编码的字符串，分割成64k大小的chunks

string ::= 's' b1 b0 <utf8-data> string #非末尾chunk

 ::= 'S' b1 b0 <utf8-data> #长度范围为(0-65535)的字符串

 ::=[x00-x1f] <utf8-data> #长度范围为(0-31) 的字符串

 #map/list 的类型（针对面向对象语言)

type ::= 't' b1 b0 <type-string> #类型名称

 ::= x75 int #类型引用值（用整数表示）

 #main production

value ::=null

 ::= binary

 ::= boolean

 ::= date

 ::= double

 ::= int

 ::= list

 ::= long

 ::= map

 ::= class-def value

 ::= ref

 ::= string

**4. 序列化**

Hessian的对象序列化机制包含8种基本类型:

1.         原始二进制数据

2.         boolean

3.         64-bit date

4.         64-bit double

5.         32-bit int

6.         64-bit long

7.         null

8.         UTF8编码的string

另外包括3种递归类型：

1.         list for lists and arrays

2.         map for maps and dictionaries

3.         object for objects

最后，它还包含一个特殊的类型：

1.         ref 用来表示对共享对象的引用.

Hessian 2.0有3个内部的引用表:

1. 一个object/list 引用表.

2.         一个类型定义(class definition)引用表.

3.         一个type(class name)引用表.

**4.1. 二进制数据**

二进制数据语法

binary ::= b b1 b0 <binary-data> binary

 ::= B b1 b0 <binary-data>

 ::= [x20-x2f] <binary-data>

二进制数据被分割成chunk. 十六进制数x42('B')标识最后一个chunk,x62('b')标识普通的chunk.

每个chunk有一个16-bit的长度值.

len = 256\*b1 + b0

**4.1.1. 压缩格式：短二进制**

长度小于15的二进制数据只需要用单个十六进制数字来表示[x20-x2f].

len = code - 0x20

**4.1.2. Binary实例**

x20 #零长度的二进制数据

x23 x01 x02 x03 #长度为3的数据

B x10 x00 .... #4k大小的final chunk

b x04 x00 .... #1k大小的non-final chunk

**4.2. boolean**

Boolean语法

boolean ::= T

 ::= F

用16进制'F'来表示false,用'T'表示true.

4.2.1. Boolean实例

T # true

F # false

**4.3. date**

date ::=d b7 b6 b5 b4 b3 b2 b1 b0

Date采用64-bit来表示距1970 00:00H, UTC以来经过的milliseconds.

**4.3.1. Date实例**

D x00 x00 xd0 x4b x92 x84 xb8 #2:51:31 May 8, 1998 UTC

**4.4. double**

double ::= D b7 b6 b5 b4 b3 b2 b1 b0

 ::= x67

 ::= x68

 ::= x69 b0

 ::= x6a b1 b0

 ::= x6b b3 b2 b1 b0

符合IEEE标准的 64-bit浮点数.

**4.4.1. 压缩格式：double表示的0**

0.0用十六进制x67来表示 （对应ascii中字符g的ascii值）

**4.4.2. 压缩格式：double 表示的1**

1.0用十六进制x68来表示

**4.4.3. 压缩格式：单字节double**

介于-128.0和127.0之间的无小数位的double型可以用两个十六进制来表示(如x3b表示的),也即相当于一个byte值转换成double:

value = (double)b0

**4.4.4. 压缩格式：short型double**

介于-32768.0和32767.0之间的无小数位的double型可以用3个十六进制数来表示，也即相当于一个short值转换成double:

 value=(double)(256\*b1 + b0)

**4.4.5. float型double**

和32-bit float型等价的double能够用4个十六进制的 float来表示.

**4.4.6. Double实例**

x67 # 0.0

x68 # 1.0

x69 x00 # 0.0

x69 x80 # -128.0

x69 xff # 127.0

x70 x00 x00 # 0.0

x70 x80 x00 # -32768.0

x70 xff xff # 32767.0

D x40 x28 x80 x00 x00 x00 x00 x00 # 12.25

**4.5. int**

int ::= ’I’ b3 b2 b1 b0

 ::= [x80-xbf]

 ::= [xc0-xcf] b0

::= [xd0-xd7] b1 b0

32-bit的有符号整型. 一个整型由跟随在x49(‘I’)之后的4个大端序(big-endian)排位的十六进制数来表示。

value = (b3<<24) + (b2<<16) + (b1<<8) + b0;

**4.5.1. 单字节整型**

介于-16和47之间的整型可以用单个字节来表示，用十六进制来表示范围为x80到xbf.

value = code – 0x90 # 这里是0x90, 如果code=x80，则value = x80 – x90 = -16.:-)

**4.5.2. 双字节整型**

介于-2048和2047之间的整型可以用两个字节来表示，头字节的范围从xc0到xcf.

value = ((code – 0xc8)<<8) + b0;

**4.5.3. 三字节整型**

介于- 262144和262143之间的整型可以用三个字节来表示，头字节的范围从xd0到xd7.

**4.5.4. 整型实例**

x90 # 0

x80 # -16

xbf # 47

xc8 x00 # 0

xc0 x00 # -2048

xc7 x00 # -256

xcf xff # 2047

xd4 x00 x00 # 0

xd0 x00 x00 # -262144

xd7 xff xff # 262143

I x00 x00 x00 x00 # 0

I x00 x00 x01 x2c # 300

**4.6. list**

list ::= V type? length? value\* z

 ::=V int int value\*

一个有序列表.每个list都包含一个type字符串，长度length和一个值列表，以十六进制x7a(‘z’)作为结尾。Type可以是任意的UTF-8编码的字符串。Length指定了list值列表的长度。

list的每个值都被添加到一个引用列表中，这样，所有list中的相同条目都共享同一份引用以节省空间。

Any parser expecting a list must also accept a null or a shared ref.

Type的有效取值在文档中并没有详细指定，这依赖于特定的应用. 比如， 在一个由静态类型语言实现的server所暴露的Hessian接口可以使用类型信息来实例化特定的数组类型，反之，在一个由动态类型语言(e.g.: python)实现的server中，将会忽略类型信息。

**4.6.1. 压缩格式: repeated list**

Hessian2.0 制定了一个格式紧凑的list，其中list元素类型type和元素个数length都用整型来编码，其中类型type是对先前定义的原始数据类型的引用。

**4.6.2. List实例**

强类型int数组的序列化: int[] = {0, 1}

V

 t x00 x04 [int # int[] 类型的编码

 x6e x02 # length = 2

 x90 # 整数 0

 x91 # 整数 1

 z

匿名变长list = {0, “foobar”}

V

 t x00 x04 [int # int[] 类型编码

 x6e x02 # length = 2

 x90 # 整数0

 x91 # 整数1

 z

**Repeated list类型**

V

 t x00 x04 [int # int[]类型编码

 x63 x02 # length=2

 x90 # 整数 0

 x91 # 整数 1

 z

V

 x91 # int[]的类型引用 (integer #1)

 x92 # length = 2

 x92 # 整数2

 x93 # 整数3

**4.7. long**

long ::= L b7 b6 b5 b4 b3 b2 b1 b0

 ::= [xd8-xef]

 ::= [xf0-xff] b0

 ::= [x38-x3f] b1 b0

 ::= x77 b3 b2 b1 b0

一个64-bit的有符号整数. 起头由十六进制x4c(‘L’)标识, 后面为8字节的大端（big-endian）序的整数。

**4.7.1. 压缩格式: 单字节long**

介于-8和15之间的long由单个字节表示，范围为xd8到xef.

value = (code – 0xe0)

**4.7.2. 压缩格式: 双字节long**

介于-2048和2047之间的long由两个字节标识, 起头字节的取值范围为xf0到xff.

value = ((code – 0xf8)<<8) + b0

**4.7.3. 压缩格式: 3字节long**

介于-262144和262143之间的long由3个字节编码，起头字节的取值范围为x38到x3f.

value = ((code – 0x3c)<<16) + (b1<<8) + b0

**4.7.4. 压缩格式: 四字节long**

可以用32-bit的整数来标识的long在这里需要用5个字节作编码,起头字节由x77标识.

value = (b3<<24) + (b2<<16) + (b1<<8) + b0

**4.7.5. long实例**

xe0 # 0

xd8 # -8

xef # 15

xf8 x00 # 0

xf0 x00 # -2048

xf7 x00 # -256

xff xff # 2047

x3c x00 x00 # 0

x38 x00 x00 # -262144

x3f xff xff # 262143

x77 x00 x00 x00 x00 # 0

x77 x00 x00 x01 x2c # 300

L x00 x00 x00 x00 x00 x00 x01 x2c # 300

**4.8. map**

map ::= M type? (value value)\* z

用来表示序列化map和对象. Type字段用来表示map类型，type可能为空（例如在length为0的情况下）。如果类型未指定，则由解析器来负责选择类型。对于对象而言,不识别的key将会被忽略.

所有map元素也被存入一个引用列表. 在解析map时，可以同时支持空类型和引用类型。

类型由服务具体来进行选择。

**4.8.1. Map实例**

Map map = new HashMap();

map.put(new Integer(1), “fee”);

map.put(new Integer(16), “fie”);

map.put(new Integer(256), “foe”);

M

 x91 # 1

 x03 fee # “fee”

 xa0 # 16

 x03 fie # “fie”

 xb9 x00 # 256

 x03 foe # “foe”

z

由java对象表示的Map对象:

Public class Car implements Serializable{

 String color = “aquamarine”

 String model = “Beetle”;

 Int mileage = 65536;

}

M

 t x00 x13 com.caucho.test.Car #type

 x05 color # color field

 x0a aquamarine

 x05 model # model field

 x06 Beetle

 x07 mileage #mileage field

 I x00 x01 x00 x00

 z

**4.9. null**

null ::= N

null表示一个空指针。字符’N’用来标识null值。

**4.10. 对象(object)**

object ::= ‘o’ int value\*

class-def ::=’O’ type int string\*

**4.10.1. 压缩格式: class定义**

Hessian 2.0制定了一个紧凑的对象格式，其中字段名只需要序列化一次。对于对象而言，仅需要按次序地序列化这些字段的取值.

类定义包括类型字符串,字段个数,所有的字段名. 类定义被存放在一个对象定义表中,每个类定义由一个唯一整数作为key标识，之后被对象实例通过key来引用。

**4.10.2. 压缩格式: 对象实例**

对象实例基于先前定义来创建一个新的对象实例. 类定义通过一个整型key在类定义表中进行查询。

**4.10.3. 对象实例**

对象序列化

Class Car{

 String color;

 String model;

}

out.wirteObject(new Car(“red”, “corvette”));

out.writeObject(new Car(“green”, “civic”));

O # 类型定义 (假定在类型表中对应key为 0)

 t x00 x0b example.Car # 类型为example.Car

 x92 # 两个字段

 x05 color # color字段名

 x05 model # model字段名

o

 x90 # 对象定义 (类型引用key=0)

 x03 red # color字段取值

 x08 corvette # model字段取值

o

 x90 # 对象定义 (类型引用key=0)

 x05 green # color字段取值

 x05 civic # model字段取值

enum Color{

 RED,

 GREEN,

 BLUE

}

out.writeObject(Color.RED);

out.writeObject(Color.GREEN);

out.writeObject(Color.BLUE);

out.writeObject(Color.GREEN);

O # 类型定义 (假定在类型表中对应key为 0)

 t x00 x0b example.Color # 类型为example.Color

 x91 # 一个字段

 x04 name # 枚举字段为”name”

o # 对象 #0

 x90 # 对象定义 (类型引用key=0)

 x03 RED # RED

o # 对象 #1

 x90 # 对象定义 (类型引用key=0)

 x05 GREEN # GREEN

o # 对象 #2

 x90 # 对象定义 (类型引用key=0)

 x05 BLUE # BLUE

x4a x01 #对象引用 #1

**4.11. 引用(ref)**

ref ::= R b3 b2 b1 b0

 ::= x4a b0

 ::= x4b b1 b0

以一个整数作为key,引用之前所定义的list, map或者对象实例. 对于从输入流中读取出来的每个list, map或对象, 都在流中按位置次序作为标识, i.e. 第一个list或者map为'0', 下一个为'1'等等. 之后的引用能够使用先前的对象. Writers MAY生成引用. Parsers MUST 识别引用.

ref can refer to incompletely-read items. For example, a circular linked-list will refer to the first link before the entire list has been read.

A possible implementation would add each map, list, and object to an array as it is read. The ref will return the corresponding value from the array. To support circular structures, the implementation would store the map, list or object immediately, before filling in the contents.

Each map or list is stored into an array as it is parsed. ref selects one of the stored objects. The first object is numbered '0'.

**4.11.1. 压缩格式: 双字节引用**

介于0和255之间的引用编号可以用两个字节来编码

Value = b0

**4.11.2. 压缩格式: 三字节引用**

介于0和255之间的引用编号可以用三个字节来编码

Value = (b1<<8) + b0

**4.11.3. 引用实例**

循环链表

list = new LinkedList();

list.data = 1;

list.tail = list;

O

 X9a LinkedList # 类型定义 (假定在类型表中对应key为 0)

 X92

 X04 head

 X04 tail

o x90 # 类引用 #0

 x91 # data = 1

 x4b x00 # 下一个条目引用到自身， ref #0

**4.12. string**
string ::= s b1 b0 <utf8-data> string

 ::= s b1 b0 <utf8-data>

 ::= [x00-x1f] <utf8-data>

16-bit字符，UTF-8编码的字符串。字符串分块(chunk)编码. x53(‘S’)起头来标识最后一个chunk, x73(‘s’)起头标识非最终chunk. 每个chunk有一个16-bit的长度值字段。

length用来标识字符串的长度，而非字节数量.

String chunks may not split surrogate pairs.

**4.12.1. 压缩格式: 短字符串**

长度小于32的字符串可以用单字节长的length来标识，范围为[x00-x1f].

value = code

**4.12.2 字符串实例**

x00 # “”, 空字符串

x05 hello # “hello”

x01 xc3 x83 # “\u00c3”

S x00 x05 hello # 长字符串格式的“hello”

s x00 x07 hello, # “hello, world”被分割成两个chunk

 X05 world

**4.13. 类型(type)**

type ::= ‘t’ b1 b0 <type-string>

 ::= x4a b0

在面向对象语言中，一个map或list可能包含一个type属性以标识map或list的类型名称。

每个type都被加入到类型表(type map)中，以供将来被引用。

**4.14. 压缩格式: 类型引用**

Repeated type strings MAY use the type map (type reference) to refer to a previously used type. The type reference is zero-based over all the types encountered during parsing.

**5. 引用表(Reference Map)**

Hessian 2.0包含3个内建的引用表：

1.         一个map/object/list引用表.

2.         一个类型定义表(class definition map).

3.         一个type(class name)表

值引用表使得hessian支持arbitrary graphs，还有递归和循环数据结构。

class和type表通过消除重复字符串而提高了hessian的效率。

**5.1. 值引用**

Hessian supports arbitrary graphs by adding list (list), object (object), and map (map) as it encounters them in the bytecode stream。

解析器必须在引用表中存储每个list, 对象和map。

被存储的对象能够被引用。

**5.2. class引用**

每个类型定义被自动添加到类型表(class-map)中。解析器必须在初次使用一个类型时把它添加到类型表中，之后对象实例将通过引用来追踪自身的类型。

**5.3. type引用**

map和list的值类型字符串被存储在type map中。解析器必须在初次碰到一个type字符串时把它加入到type map中。

**6. 字节码映射表(Bytecode map)**

Hessian被制定成一个字节码协议。Hessian解析器本质上是一段针对头字节的选择分支语句.

字节编码

x00 - x1f # utf-8字符串，长度范围 0-32

x20 - x2f # 二进制数据，长度范围 0-16

x30 - x37 # 保留

x38 - x3f # 长整型long 范围从-x40000 到 x3ffff

x40 - x41 # 保留

x42 # 8-bit 二进制数据，表示末尾chunk('B')

x43 # 保留('C' streaming call)

x44 # 64-bit IEEE 规范编码的双精度浮点double ('D')

x45 # 保留('E' envelope)

x46 # boolean false ('F')

x47 # 保留

x48 # 保留 ('H' header)

x49 # 32-bit有符号整型signed integer ('I')

x4a # 引用(ref)，范围为1-256th

x4b # 引用(ref)，范围为1-65536th

x4c # 64-bit有符号长整型long integer ('L')

x4d # 具有可选类型的map ('M')

x4e # null ('N')

x4f # 类型定义('O')

x50 # 保留('P' streaming message/post)

x51 # 保留

x52 # 引用(ref)，取值范围对应于整型int ('R')

x53 # utf-8字符串，末尾chunk ('S')

x54 # boolean true ('T')

x55 # 保留

x56 # list/vector ('V')

x57 - x62 # 保留

x62 # 8-bit二进制数据，非末尾chunk ('b')

x63 # 保留 ('c' call for RPC)

x64 # UTC time encoded as 64-bit long milliseconds since epoch ('d')

x65 # 保留

x66 # 保留('f' for fault for RPC)

x67 # double 0.0

x68 # double 1.0

x69 # double represented as byte (-128.0 to 127.0)

x6a # double represented as short (-32768.0 to 327676.0)

x6b # double represented as float

x6c # list/vector length ('l')

x6d # 保留 ('m' method for RPC call)

x6e # list/vector compact length

x6f # 对象实例('o')

x70 # 保留 ('p' - message/post)

x71 # 保留

x72 # 保留('r' reply for message/RPC)

x73 # utf-8字符串，非末尾chunk ('s')

x74 # map/list type ('t')

x75 # type-ref

x76 # 压缩格式的vector ('v')

x77 # 以32-bit整型编码的long

x78 - x79 # 保留

x7a # list/map 终止符('z')

x7b - x7f # 保留

x80 - xbf # 单字节压缩格式的整型int(-x10 to x3f, x90 is 0)

xc0 - xcf # 双字节压缩格式的整型int(-x800 to x3ff)

xd0 - xd7 # 三字节压缩格式的整型int(-x40000 to x3ffff)

xd8 - xef # 单字节压缩格式的长整型long(-x8 to x10, xe0 is 0)

xf0 - xff # 双字节压缩格式的长整型long (-x800 to x3ff, xf8 is 0)