JSON数据乱码问题

背景

程序员一提到编码应该都不陌生，像gbk、utf-8、ascii等这些编码更是经常在用，但时不时也会出个乱码问题，解决这个问题的方法大部分都是先google和baidu一下，最后可能在某个犄角旮旯里找到一点信息，然后就机械的按部就班的模仿下来,结果问题可能真就迎刃而解了,然后就草草了事,下回遇到相似的问题,可能又是重复上面的过程。很少有人有耐心去花精力弄明白这写问题的根本原因，以及解决这些问题的原理是什么。这篇文章就是通过一个实际案例,试着去讲清楚什么是编码,乱码又是怎么产生的,以及如何解决。该案例是从lua_cjson.c这个库开始的,对这个库不熟悉也没关系,也不需要熟悉它,我们只是借用它来说明乱码问题,只需要跟着文章的思路走就可以。

前段时间同事在作一个新项目的时候用到了lua_cjson.c这个库(以下简称cjson)，将json串转换成LUA本地的数据结构,但是在使用的过程中出现了中文乱码问题，奇怪的是只有那么几个字是乱码，这其中就包括”朵”字，其他字一切正常。经了解json串用的是GBK编码，那问题就来了，为什么用gbk编码会出现这个问题，原因是什么？又应该怎么解决这个问题？

要解释清楚这个问题，首先我们来看看json串都有哪些要求。

JSON规范

json全称JavaScript Object Notion是结构化数据序列化的一个文本，可以描述四种基本类型(strings,numbers,booleans and null)和两种结构类型(objects and arrays)。

RFC4627中有这样一段话

A string is a sequence of zero or more Unicode characters.

字符串有零个或多个*unicode*字符序列组成.

在这里稍微解释下什么是unicode字符。我们都知道ascii字符有字母、数字等,但是他收录的字只有一百多个。比如汉字就不是ascii字符，但是unicode收录了汉字,所以汉字可以是unicode字符。

这里要说明的是unicode字符其实就是一些符号。

现在另一个问题出来了，在json文本中应该怎么表示这些字符。

在规范的Encoding片段是这样说的

JSON text SHALL be encoded in Unicode. The default encoding is UTF-8。

*JSON*文本*SHALL*把*unicode*字符编码。默认使用*utf-8*编码。

我们看到在这里用到了SHALL[RFC2119]这个关键字，也就是说字符必须被编码后才能作为JSON串使用。而且默认使用utf-8编码。

如何判断使用的是那种unicode编码呢?

Since the first two characters of a JSON text will always be ASCII characters[RFC0020], it is possible to determine whether an octet stream is UTF-8、UTF-16(BE or LE), or UTF-32(BE or LE)by looking at the pattern of nulls in the first four octets.

由于*json*文本的前两个字符*(*注意这里说的是字符,不是字节*)*一定是*ASCII*字符,因此可以从一个字节

流的前四个字节*(*注意是字节*)*中判断出该字节流是*UTF-8*、*UTF-16(BE or LE)*、*or UTF-32(BE or LE)*编码。

*00 00 00 xx UTF-32BE (u32*编码大端*)*

*xx 00 00 00 UTF-32LE (u32*编码小端*)*

*00 xx 00 xx UTF-16BE (u16*编码大端*)*

*xx 00 xx 00 UTF-16LE (u16*编码小端*)*

*xx xx xx xx UTF-8 (utf-8*编码*)*

ps:

*u32*用*32*位的*4*字节整数表示一个字符；

*u16*用*16*位的*2*字节整数表示一个字符,如果*2*字节表示不了,就用连续两个*16*位的*2*字节整数表示，所以就会出现*u16*编码中有*4*个字节表示一个字符的情况，和*u32*的四字节不一

样的是,该字符在*u16*中的前两个字节和后两个字节之间不会有字序的问题。

*utf-8*用多个*8*位的*1*字节序列来表示一个字符,所以没有字序的问题.

截止到现在我们没有看到任何关于可以使用GBK编码的信息,难道json文本就不能用gbk编码吗,如果真的不能用的话，那为什么cjson不是把所有的gbk编码解释称乱码,而是只有某几个字是乱码.

在规范中对json解析器有这样一段描述:

A JSON parser transforms a JSON text into another representation.

A JSON parser MUST accept all texts that conform to the JSON grammar.

A JSON parser MAY accept non-JSON forms or extensions.

json解析器可以将一个json文本转换成其他表示方式。

json解析器MUST接受所有符合json语法的文本.

json解析器MAY接受非json形式或扩展的文本.

乱码的原因

从规范对对解析器的描述可以看到,规范并没有要求解析器必须对文本的编码方式做校验,而且解析器也可以有选择的去接受非json形式的文本。

现在我们再来看看cjson解析器是如何做的，在cjson开头的注释中说了这么一句话:

Invalid UTF-8 characters are not detected and will be passed untouched。

If required, UTF-8 error checking should be done outside this library。

发现无效的*UTF-8*编码会直接放过,如果有必要对*UTF-8*编码的检查应该在该库的之外。

说的很清楚,对非utf8编码直接放过，不做任何检查,所以用gbk编码不符合规范,但又可以被解析的答案就出来了。那”朵”等这些字的乱码问题又是怎么回事? 我们现在看看cjson对规范中的另外两个编码utf16、utf32是如何做的，然后再说乱码问题.

在cjson解析方法的开始处是这么做的:

```
/* Detect Unicode other than UTF-8(see RFC 4627, Sec 3)
```

```
*
* CJSON can support any simple data type, hence only the first
* character is guaranteed to be ASCII (at worst:'"'). This is
* still enough to detect whether the wrong encoding is in use.
*/
if (json_len >=2 && (!json.data[0] || !json.data[1]))
luaL_error(1,"JSON parser does not support UTF-16 or UTF-32");
```

前面我们说过一个json串的前两个字符一定是ascii字符,也就是说一个json串至少也的有两个字节.所以这段代码首先判断json串的长度是不是大于等2,然后根据串的前两个字节的值,是否有零来判断该文本是否是非utf-8编码。结果已经看到了，人家不支持规范上说的u16和u32编码.

现在我们就来看看"朵"这个子是如何变成乱码的，经过对cjson源码的分析得知,cjson在处理字节流的时候当遇见'\'反斜杠时会猜测后一个字节应该是要被转义的字符,比如\b、\r之类的字符,如果是就放行,如果不是,cjson就认为这不是一个正确的json格式,就会把这个字节给干掉,所以本来用两个字节表示的汉子就硬生生的给掰弯了。

那"朵"字跟'\'反斜杠又有什么关系? 查询这两字符在编码中的表示得出:

"朵" 0x965C

"\" 0x5C

这样我们就看到"朵"字的低位字节和"\"字符相同,都是0x5C,如果这时候"朵"字后边不是b、r之类的可以被转移ascii字符,cjson就会把这个字节和紧跟其后的一个字节抹掉,所以乱码就产生了。

那我们应该怎么解决这个问题,让cjson可以顺利的支持gbk编码呢,首先我们看看gbk编码是怎么回事,为什么会出现低位字节和ascii冲突的问题.

GB_编码系列

先来了解一下GB系列的编码范围问题:

GB2312(1980)共收录7445个字符，6763个汉字和682个其他字符。

每个汉字及符号用两个字节表示,为了跟ascii兼容,处理程序使用EUC存储方法。

汉字的编码范围

高字节*: 0xB0 – 0xF7,*

低字节*: 0xA1 – 0xFE,*

占用72*94=6768,0xD7FA – 0xD7FE未使用。

GBK共收录21886个字符,采用一字节和双字节编码。

单字节表示范围

*8*位*: 0x0 – 0x7F*

双字节表示范围

高字节*: 0x81 – 0xFE*

低字节*: 0x40 – 0x7E*、*0x80 – 0xFE*

GB18030收录70244个汉字,采用1、2、4字节编码。

单字节范围

*8*位*: 0x0 – 0x7F*

双字节范围

高字节*: 0x81 – 0xFE*

低字节*: 0x40 – 0xFE*

四字节范围

第一字节*:0x81 – 0xFE*

第二字节*:0x30 – 0x39*

第三字节*:0x81 – 0xFE*

第四字节*:0x30 – 0x39*

由于GB类的编码都是向下兼容的，这里就有一个问题，因为GB2312的两个字节的高位都是1,符合这个条件的码位只有128*128=16384个。GBK和GB18030都大于这个数,所以为了兼容,我们从上面的编码范围看到,这两个编码都用到了低位字节的最高位可以为0的情况。

最终得出的结论就是,在GBK编码中只要该字符是两个字节表示,并且低位字节是0x5C的字符都会被cjson弄成乱码.

解决方案:

1) 不要使用gbk编码,将你的字符串转换成utf-8编码.

2) 对cjson源码稍微做个改动,就是在每个字节到来之前先判断该字节是否大于127,如果大于则将该字节个随后的一个字节放过,否则交给cjson去处理。

About Latest Posts

Deyimsf