<https://blog.csdn.net/yzhang6_10/article/details/51615089>

* **简介**

由于C语言汇总，没有直接的字典，字符串数组等数据结构，所以要借助结构体定义，处理json。

**JSON是一种轻量级的数据交换格式**。JSON采用完全独立与语言的文本格式，易于人阅读和编写。同时也易于机器解析和生成。它是基于JavaScript，Programming Language，Standard ECMA-262 3rd Edition -December 1999的一个子集。JSON采用完全独立于语言的文本格式，但是也使用了类似于C语言家族的习惯（如C，C++，C++，JavaScript，Perl，Python等）。这些特性使用JSON成为理想的数据交换语言。

**JSON作用：在数据传输时能够更好地提取出需要的数据，可以用于客户端和服务器端的数据交互。**

JSON建构与两种结构：

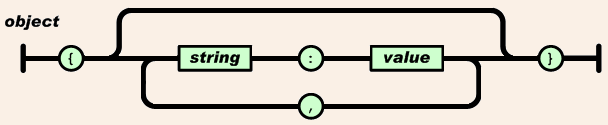
* + “名称/值”对的集合。不同的语言中，它被理解为对象（Object），记录（Record），结构（struct），字典（dictionary），哈希表（hash table），有键列表（Key list），或者关联数组（Associative array）。
  + 值的有序列表（An ordered list of values）。在大部分语言中，它被理解为数组（array）。

**JSON的结构可以理解成无序的、可嵌套的key-value键值对集合**，这些key-value键值对是以结构体或数组的形式来组织的。同一级的key-value键值对之间是用以个“，”（逗号）隔开，每个key-value键值对是由一个key后面紧接一个“：”（冒号），冒号后面是这个key对应的value。Key是一个word，由大小写字母、下划线及数字组成，可以由双引号封闭，也可以不加双引号；而value值的取值为：**字符串（string），数值（number），true，false，null，对象（object）或者数组（array）**。这些结构可以嵌套。

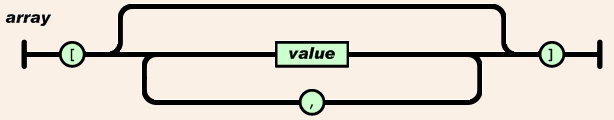
**实质：JSON是一种信息交换格式，而cJSON就是对JSON格式的字符串进行构建和解析的一个C语言函数库。此外，cJSON作为JSON格式的解析库，其主要功能就是构建和解析JSON格式。**

JSON具有的形式如下：

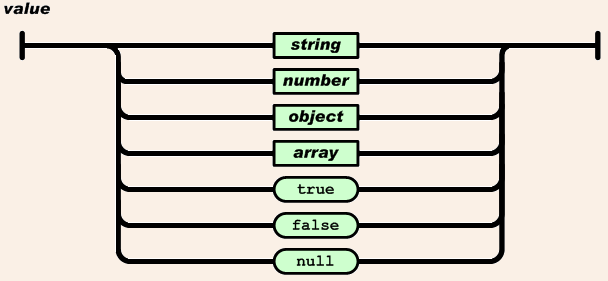
对象是一个无序的“名称/值”对集合。一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“：”（冒号）；“‘名称/值’”对之间使用“，”（逗号）分割。其具体形式如下图：



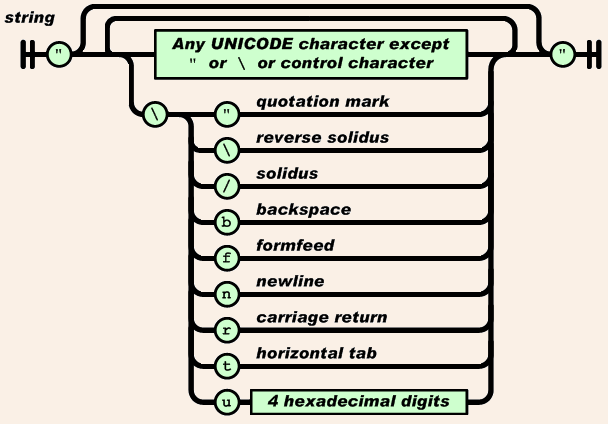
数值是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间使用“，”（逗号）分割。其具体形式如下图：



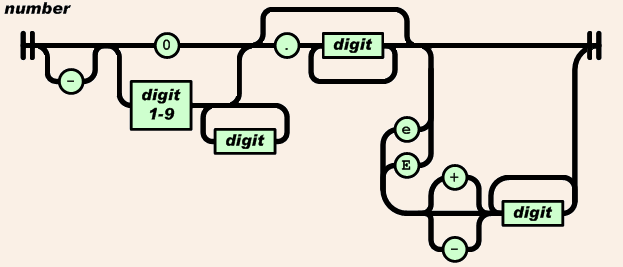
值（value）可以是双引号括起来的字符串（string）、数值（number）、true、false、null、对象（object）或者数组（array）。这些结构可以嵌套。其具体形式如下：



字符串（string）是由双引号包围的任意数量Unicode字符的集合，使用反斜线转义。一个字符（character）即一个单独的字符馋（character string）。字符串（string）与C或者Java的字符串非常相似。其具体形式如下：



数值（number）也与C或者Java的数值非常相似。除去未曾使用的八进制与十六进制格式。除去一些编码细节。其具体形式如下：



JSON格式举例如下：

对象: 名称/值的集合

例:{"firstName":"Tom"}

数组: 值的序列

例:[310, null, 0.231, -2.3E+5]

字符串:

例:"cJSON"

数字:

例:500

布尔值:

例:true false

* **cJSON源码分析**

1）cJSON源码下载，网址为： <http://sourceforge.net/projects/cjson/>。

2）解压后，主要参看的源码文件为cJSON.h和sJSON.c和test.c，其中test.c为测试函数。

由于cJSON为JSON格式的解析库，故主要功能是构建和解析JSON格式。其中的结构体，函数定义实现等都是围绕这两个函数实现。下面将对其源码进行分析。

JSON的内存结构不是树，像广义表，可以认为是有层次的双向链表。

cJSON中的重要接口函数如下：

解析函数

cJSON \* cJSON\_Parse(const char \*value);

打印函数

char \* cJSON\_Print(cJSON \* item);

删除函数

void cJSON\_Delete(cJSON \* c);

构造函数

create系列和add系列

解析字符串

char \*parse\_string(cJSON\*item,const char \*str)

解析数字

char \*parse\_number(cJSON \*item,const char \*num)

解析数组

char \*parse\_array(cJSON \*item,const char \*value)

解析对象

char \*parse\_object(cJSON \*item,const char \*value)

......

cJSON程序中的细节点如下：

* + 大量宏替换
  + 大量静态函数
  + 错误处理机制
  + 字符串处理时存在utf16转utf9，编码转换
  + 用函数指针封装malloc，free，方便用于处理，比如在申请后初始化，或者释放前进行一些处理等。

CJSON的节点结构体如下：

// JSON的一个value的结构体

typedef struct cJSON

{

struct cJSON \*next,\*prev; // 同一级的元素使用双向列表存储

struct cJSON \*child; // 如果是一个object或array的话，child为第一个儿子的指针

int type; // value的类型

char \*valuestring; // 如果这个value是字符串类型，则此处为字符串值

int valueint; // 如果是数字的话，整数值

double valuedouble; // 如果是数字的话，读点数值

char \*string; // 如果是对象的key-value元素的话，key值

} cJSON;

// JSON的类型

#define cJSON\_False 0

#define cJSON\_True 1

#define cJSON\_NULL 2

#define cJSON\_Number 3

#define cJSON\_String 4

#define cJSON\_Array 5

#define cJSON\_Object 6

#define cJSON\_IsReference 256

#define cJSON\_StringIsConst 512

cJSON中的内存管理使用了HOOK技术，主要是为了方便使用者自己定义内存管理函数，即用户自定义的malloc和free。下面对其内存管理相关程序分析。

// json内存管理

// 为方便用户自由的管理内存，其使用了Hook技术让使用者可以自定义内存管理函数

typedef struct cJSON\_Hooks

{

void \*(\*malloc\_fn)(size\_t sz);

void (\*free\_fn)(void \*ptr);

} cJSON\_Hooks;

// 对cJSON提供的分配，再分配，释放内存初始化函数

extern void cJSON\_InitHooks(cJSON\_Hooks\* hooks);

// 默认将分配和释放空间函数指针指向malloc和free

static void \*(\*cJSON\_malloc)(size\_t sz) = malloc;

static void (\*cJSON\_free)(void \*ptr) = free;

// 其使用Hook技术来让使用者可以自定义内存管理函数。其中默认系统使用的内存分配和释放函数是malloc

// 和free函数，利用cJSON\_InitHooks函数可以替换成用户自定义的malloc和free函数。

void cJSON\_InitHooks(cJSON\_Hooks\* hooks)

{

// 如果未定义，则使用默认的malloc和free函数

if (!hooks) { /\* Reset hooks \*/

cJSON\_malloc = malloc;

cJSON\_free = free;

return;

}

// 定义了，则使用用户自定义的malloc和free函数

cJSON\_malloc = (hooks->malloc\_fn)?hooks->malloc\_fn:malloc;

cJSON\_free = (hooks->free\_fn)?hooks->free\_fn:free;

}

构建JSON格式数据，首先调用CJSON\_CreateObject()函数，返回一个类型为cJSON\_Object的cJSON的结构体，这其中调用了CJSON\_CreateNULL()、CJSON\_CreateTrue()、…、创建不同类型数据的CJSON结构其。在构建过程中，调用CJSON\_New\_Item创建对应节点信息；然后调用cJSON\_AddItemToObject()并结合不同的对象类型增加节点名称和子节点。然后在其中调用cJSON\_AddItemToArray()函数来添加信息，此函数中判断对象孩子结点是否为NULL，如果是NULL，则直接插入，否则找到最后一个孩子，调用suffix\_object()函数添加到双向链表的尾部。具体程序如下。

// 利用宏函数来快速增加cJSON相关节点信息

// 创建一个string值为name的cJSON\_Null节点，并添加到object

#define cJSON\_AddNullToObject(object,name) cJSON\_AddItemToObject(object, name, cJSON\_CreateNull())

// 创建一个string值为name的cJSON\_True节点，并添加到object

#define cJSON\_AddTrueToObject(object,name) cJSON\_AddItemToObject(object, name, cJSON\_CreateTrue())

// 创建一个string值为name的cJSON\_False节点，并添加到object

#define cJSON\_AddFalseToObject(object,name) cJSON\_AddItemToObject(object, name, cJSON\_CreateFalse())

// 创建一个string值为name的cJSON\_CreateBool节点，并添加到object。b非0为cJSON\_True，0为cJSON\_False。

#define cJSON\_AddBoolToObject(object,name,b) cJSON\_AddItemToObject(object, name, cJSON\_CreateBool(b))

// 创建一个string值为name，valuedouble为n，valueint为（int）n的cJSON\_Number节点，并添加到object。

#define cJSON\_AddNumberToObject(object,name,n) cJSON\_AddItemToObject(object, name, cJSON\_CreateNumber(n))

// 创建一个string值为name，valuestring为s的cJSON\_String节点，并添加到object。

#define cJSON\_AddStringToObject(object,name,s) cJSON\_AddItemToObject(object, name, cJSON\_CreateString(s))

// 函数解析

// 输入参数无

// 返回值：指向一个cJSON\_Object类型节点的指针

// 创建一个cJSON节点，并设置节点类型无cJSON\_Object

extern cJSON \*cJSON\_CreateObject(void);

cJSON \*cJSON\_CreateObject(void)

{

// 创建节点

cJSON \*item=cJSON\_New\_Item();

if(item)

item->type=cJSON\_Object;

return item;

}

// 创建value节点

static cJSON \*cJSON\_New\_Item(void)

{

// 分配空间

cJSON\* node = (cJSON\*)cJSON\_malloc(sizeof(cJSON));

// 分配成功后，初始化为0

if (node) memset(node,0,sizeof(cJSON));

return node;

}

// object(cJSON \*)：被添加节点的节点

// string(char \*)：要添加节点的名称

// item(cJSON \*)：要添加节点

// 返回值无

// 函数功能：将item节点的名称设置为string。如果object节点没有子节点，就将item设置为object

// 子节点，否则将item添加到object->child链表的尾部，成为object->child的兄弟节点

extern void cJSON\_AddItemToObject(cJSON \*object,const char \*string,cJSON \*item);

// 将字符串添加进对象

void cJSON\_AddItemToObject(cJSON \*object,const char \*string,cJSON \*item)

{

if (!item)

return;

if (item->string)

cJSON\_free(item->string); // 这个儿子之前有key，先清理

item->string=cJSON\_strdup(string); // 设置key值

cJSON\_AddItemToArray(object,item); // 添加儿子

}

// 将传入的字符串复制一副本并返回新的字符串指针

static char\* cJSON\_strdup(const char\* str)

{

size\_t len;

char\* copy;

len = strlen(str) + 1;

// 分配空间

if (!(copy = (char\*)cJSON\_malloc(len)))

return 0;

// 执行复制操作

memcpy(copy,str,len);

// 返回复制的副本

return copy;

}

// 添加节点到object或array中

void cJSON\_AddItemToArray(cJSON \*array, cJSON \*item)

{

cJSON \*c=array->child;

if (!item)

return;

if (!c)

{

array->child=item; // 之前不存在儿子节点，直接添加

}

else

{

while (c && c->next) // 先找到最后一个儿子

c=c->next;

suffix\_object(c,item); // 添加儿子，c是item的兄弟节点

}

}

// array的处理

static void suffix\_object(cJSON \*prev,cJSON \*item)

{

// 两个兄弟的指针互相指向对方

prev->next=item;

item->prev=prev;

}

JSON解析数据格式时所调用的函数过程如下：

首选，调用cJSON\_Parse()函数，此函数是一个二次封装函数，其内部为cJSON\_ParseWithOpts()函数，该函数用于提取更多的解析选项，如果需要，最后返回解析结束的位置。而在上面的函数中，调用parse\_value()函数进行解析，而该函数首先创建cJSON\_NewItem()创建节点，用于存放解析的JSON结构数据，然后根据不同的选项，调用解析函数，其为parse\_string()，parse\_number()，parse\_array()，parse\_objec()等。其程序解析如下：

// cJSON解析的二次封装函数

cJSON \*cJSON\_Parse(const char \*value)

{

return cJSON\_ParseWithOpts(value,0,0);

}

// 解析对象，创建一个新的根并初始化，返回一个cJSON类型

cJSON \*cJSON\_ParseWithOpts(const char \*value,const char \*\*return\_parse\_end,int require\_null\_terminated)

{

const char \*end=0;

cJSON \*c=cJSON\_New\_Item();

ep=0;

if (!c)

return 0; /\* memory fail \*/

end=parse\_value(c,skip(value));

if (!end)

{

cJSON\_Delete(c);

return 0;

} /\* parse failure. ep is set. \*/

/\* if we require null-terminated JSON without appended garbage, skip and then check for a null terminator \*/

if (require\_null\_terminated)

{

end=skip(end);

if (\*end)

{

cJSON\_Delete(c);

ep=end;

return 0;

}

}

if (return\_parse\_end)

\*return\_parse\_end=end;

return c;

}

// 解析器核心函数

static const char \*parse\_value(cJSON \*item,const char \*value)

{

if (!value)

return 0; /\* Fail on null. \*/

if (!strncmp(value,"null",4))

{

item->type=cJSON\_NULL;

return value+4;

}

if (!strncmp(value,"false",5))

{

item->type=cJSON\_False;

return value+5;

}

if (!strncmp(value,"true",4))

{

item->type=cJSON\_True;

item->valueint=1;

return value+4;

}

if (\*value=='\"')

{

return parse\_string(item,value);

}

if (\*value=='-' || (\*value>='0' && \*value<='9'))

{

return parse\_number(item,value);

}

if (\*value=='[')

{

return parse\_array(item,value);

}

if (\*value=='{')

{

return parse\_object(item,value);

}

ep=value;

return 0; /\* failure. \*/

}

static const char \*parse\_string(cJSON \*item,const char \*str)

{

const char \*ptr=str+1;

char \*ptr2;

char \*out;

int len=0;

unsigned uc,uc2;

if (\*str!='\"') // 不是字符串情况

{

ep=str;

return 0;

} /\* not a string! \*/

while (\*ptr!='\"' && \*ptr && ++len)

if (\*ptr++ == '\\')

ptr++; // 跳出前面的引用

out=(char\*)cJSON\_malloc(len+1); /\* This is how long we need for the string, roughly. \*/

if (!out)

return 0;

ptr=str+1;

ptr2=out;

while (\*ptr!='\"' && \*ptr)

{

if (\*ptr!='\\')

\*ptr2++=\*ptr++;

else

{

ptr++;

switch (\*ptr)

{

case 'b': \*ptr2++='\b'; break;

case 'f': \*ptr2++='\f'; break;

case 'n': \*ptr2++='\n'; break;

case 'r': \*ptr2++='\r'; break;

case 't': \*ptr2++='\t'; break;

case 'u': /\* transcode utf16 to utf8. \*/

uc=parse\_hex4(ptr+1);

ptr+=4; /\* get the unicode char. \*/

if ((uc>=0xDC00 && uc<=0xDFFF) || uc==0)

break; /\* check for invalid. \*/

if (uc>=0xD800 && uc<=0xDBFF) /\* UTF16 surrogate pairs. \*/

{

if (ptr[1]!='\\' || ptr[2]!='u')

break; /\* missing second-half of surrogate. \*/

uc2=parse\_hex4(ptr+3);ptr+=6;

if (uc2<0xDC00 || uc2>0xDFFF)

break; /\* invalid second-half of surrogate. \*/

uc=0x10000 + (((uc&0x3FF)<<10) | (uc2&0x3FF));

}

len=4;

if (uc<0x80)

len=1;

else if (uc<0x800)

len=2;

else if (uc<0x10000)

len=3;

ptr2+=len;

switch (len)

{

case 4: \*--ptr2 =((uc | 0x80) & 0xBF); uc >>= 6;

case 3: \*--ptr2 =((uc | 0x80) & 0xBF); uc >>= 6;

case 2: \*--ptr2 =((uc | 0x80) & 0xBF); uc >>= 6;

case 1: \*--ptr2 =(uc | firstByteMark[len]);

}

ptr2+=len;

break;

default: \*ptr2++=\*ptr; break;

}

ptr++;

}

}

\*ptr2=0;

if (\*ptr=='\"') ptr++;

item->valuestring=out;

item->type=cJSON\_String;

return ptr;

}

// 跳过这些空格

static const char \*skip(const char \*in)

{

while (in && \*in && (unsigned char)\*in<=32)

in++;

return in;

}

// parse\_number函数功能：解析数字，对输入的文本生成一个数字，并填充结果项，传入参数有两

// 个，这里先只关注num，返回值是一个字符串

static const char \*parse\_number(cJSON \*item,const char \*num)

{

double n=0,sign=1,scale=0;

int subscale=0,signsubscale=1;

if (\*num=='-') sign=-1,num++; // 判断数字是否是有符号数字

if (\*num=='0') num++; // 判断数字是否为0

if (\*num>='1' && \*num<='9')

do // 转换数字

n=(n\*10.0)+(\*num++ -'0');

while (\*num>='0' && \*num<='9');

if (\*num=='.' && num[1]>='0' && num[1]<='9') // 对小数点后边的部分进行处理，scale记录小数点后边的位数

{

num++;

do

n=(n\*10.0)+(\*num++ -'0'),scale--; // scale为小数点后的位数

while (\*num>='0' && \*num<='9');

}

if (\*num=='e' || \*num=='E') // 是否为指数，科学计数法

{

num++;

if (\*num=='+') // 判断指数后边幂的正负号

num++;

else if (\*num=='-')

signsubscale=-1,num++;

while (\*num>='0' && \*num<='9') // 处理指数后边10的幂

subscale=(subscale\*10)+(\*num++ - '0');

}

// 将字符串转换为相应的数值

n=sign\*n\*pow(10.0,(scale+subscale\*signsubscale)); /\* number = +/- number.fraction \* 10^+/- exponent \*/

item->valuedouble=n; // 将算出来的值存入缓存

item->valueint=(int)n; // 将算出来的值存入缓存

item->type=cJSON\_Number; // 目标类型为数字

return num;

}

// 从输入文本中构建array

static const char \*parse\_array(cJSON \*item,const char \*value)

{

cJSON \*child;

if (\*value!='[') {ep=value;return 0;} /\* not an array! \*/

item->type=cJSON\_Array;

value=skip(value+1);

if (\*value==']') return value+1; /\* empty array. \*/

item->child=child=cJSON\_New\_Item();

if (!item->child) return 0; /\* memory fail \*/

value=skip(parse\_value(child,skip(value))); /\* skip any spacing, get the value. \*/

if (!value) return 0;

while (\*value==',')

{

cJSON \*new\_item;

if (!(new\_item=cJSON\_New\_Item())) return 0; /\* memory fail \*/

child->next=new\_item;new\_item->prev=child;child=new\_item;

value=skip(parse\_value(child,skip(value+1)));

if (!value) return 0; /\* memory fail \*/

}

if (\*value==']') return value+1; /\* end of array \*/

ep=value;return 0; /\* malformed. \*/

}

// 从输入文本中构建object

static const char \*parse\_object(cJSON \*item,const char \*value)

{

cJSON \*child;

if (\*value!='{') {ep=value;return 0;} /\* not an object! \*/

item->type=cJSON\_Object;

value=skip(value+1);

if (\*value=='}') return value+1; /\* empty array. \*/

item->child=child=cJSON\_New\_Item();

if (!item->child) return 0;

value=skip(parse\_string(child,skip(value)));

if (!value) return 0;

child->string=child->valuestring;child->valuestring=0;

if (\*value!=':') {ep=value;return 0;} /\* fail! \*/

value=skip(parse\_value(child,skip(value+1))); /\* skip any spacing, get the value. \*/

if (!value) return 0;

while (\*value==',')

{

cJSON \*new\_item;

if (!(new\_item=cJSON\_New\_Item())) return 0; /\* memory fail \*/

child->next=new\_item;new\_item->prev=child;child=new\_item;

value=skip(parse\_string(child,skip(value+1)));

if (!value) return 0;

child->string=child->valuestring;child->valuestring=0;

if (\*value!=':') {ep=value;return 0;} /\* fail! \*/

value=skip(parse\_value(child,skip(value+1))); /\* skip any spacing, get the value. \*/

if (!value) return 0;

}

if (\*value=='}') return value+1; /\* end of array \*/

ep=value;return 0; /\* malformed. \*/

}

// 将十六进制的字符串转换为数字表示！

static unsigned parse\_hex4(const char \*str)

{

unsigned h=0;

if (\*str>='0' && \*str<='9')

h+=(\*str)-'0';

else if (\*str>='A' && \*str<='F')

h+=10+(\*str)-'A';

else if (\*str>='a' && \*str<='f')

h+=10+(\*str)-'a';

else

return 0;

h=h<<4;str++;

if (\*str>='0' && \*str<='9')

h+=(\*str)-'0';

else if (\*str>='A' && \*str<='F')

h+=10+(\*str)-'A';

else if (\*str>='a' && \*str<='f')

h+=10+(\*str)-'a';

else

return 0;

h=h<<4;str++;

if (\*str>='0' && \*str<='9')

h+=(\*str)-'0';

else if (\*str>='A' && \*str<='F')

h+=10+(\*str)-'A';

else if (\*str>='a' && \*str<='f')

h+=10+(\*str)-'a';

else

return 0;

h=h<<4;str++;

if (\*str>='0' && \*str<='9')

h+=(\*str)-'0';

else if (\*str>='A' && \*str<='F')

h+=10+(\*str)-'A';

else if (\*str>='a' && \*str<='f')

h+=10+(\*str)-'a';

else

return 0;

return h;

}

打印JSON信息

// 打印值到文本

static char \*print\_value(cJSON \*item,int depth,int fmt,printbuffer \*p)

{

char \*out=0;

if (!item) return 0;

if (p)

{

switch ((item->type)&255)

{

case cJSON\_NULL: {out=ensure(p,5); if (out) strcpy(out,"null"); break;}

case cJSON\_False: {out=ensure(p,6); if (out) strcpy(out,"false"); break;}

case cJSON\_True: {out=ensure(p,5); if (out) strcpy(out,"true"); break;}

case cJSON\_Number: out=print\_number(item,p);break;

case cJSON\_String: out=print\_string(item,p);break;

case cJSON\_Array: out=print\_array(item,depth,fmt,p);break;

case cJSON\_Object: out=print\_object(item,depth,fmt,p);break;

}

}

else

{

switch ((item->type)&255)

{

case cJSON\_NULL: out=cJSON\_strdup("null"); break;

case cJSON\_False: out=cJSON\_strdup("false");break;

case cJSON\_True: out=cJSON\_strdup("true"); break;

case cJSON\_Number: out=print\_number(item,0);break;

case cJSON\_String: out=print\_string(item,0);break;

case cJSON\_Array: out=print\_array(item,depth,fmt,0);break;

case cJSON\_Object: out=print\_object(item,depth,fmt,0);break;

}

}

return out;

}

// 打印array到文本

static char \*print\_array(cJSON \*item,int depth,int fmt,printbuffer \*p)

{

char \*\*entries;

char \*out=0,\*ptr,\*ret;int len=5;

cJSON \*child=item->child;

int numentries=0,i=0,fail=0;

size\_t tmplen=0;

/\* How many entries in the array? \*/

while (child) numentries++,child=child->next;

/\* Explicitly handle numentries==0 \*/

if (!numentries)

{

if (p) out=ensure(p,3);

else out=(char\*)cJSON\_malloc(3);

if (out) strcpy(out,"[]");

return out;

}

if (p)

{

/\* Compose the output array. \*/

i=p->offset;

ptr=ensure(p,1);if (!ptr) return 0; \*ptr='['; p->offset++;

child=item->child;

while (child && !fail)

{

print\_value(child,depth+1,fmt,p);

p->offset=update(p);

if (child->next) {len=fmt?2:1;ptr=ensure(p,len+1);if (!ptr) return 0;\*ptr++=',';if(fmt)\*ptr++=' ';\*ptr=0;p->offset+=len;}

child=child->next;

}

ptr=ensure(p,2);if (!ptr) return 0; \*ptr++=']';\*ptr=0;

out=(p->buffer)+i;

}

else

{

/\* Allocate an array to hold the values for each \*/

entries=(char\*\*)cJSON\_malloc(numentries\*sizeof(char\*));

if (!entries) return 0;

memset(entries,0,numentries\*sizeof(char\*));

/\* Retrieve all the results: \*/

child=item->child;

while (child && !fail)

{

ret=print\_value(child,depth+1,fmt,0);

entries[i++]=ret;

if (ret) len+=strlen(ret)+2+(fmt?1:0); else fail=1;

child=child->next;

}

/\* If we didn't fail, try to malloc the output string \*/

if (!fail) out=(char\*)cJSON\_malloc(len);

/\* If that fails, we fail. \*/

if (!out) fail=1;

/\* Handle failure. \*/

if (fail)

{

for (i=0;i<numentries;i++) if (entries[i]) cJSON\_free(entries[i]);

cJSON\_free(entries);

return 0;

}

/\* Compose the output array. \*/

\*out='[';

ptr=out+1;\*ptr=0;

for (i=0;i<numentries;i++)

{

tmplen=strlen(entries[i]);memcpy(ptr,entries[i],tmplen);ptr+=tmplen;

if (i!=numentries-1) {\*ptr++=',';if(fmt)\*ptr++=' ';\*ptr=0;}

cJSON\_free(entries[i]);

}

cJSON\_free(entries);

\*ptr++=']';\*ptr++=0;

}

return out;

}

// 打印object到文本中

static char \*print\_object(cJSON \*item,int depth,int fmt,printbuffer \*p)

{

char \*\*entries=0,\*\*names=0;

char \*out=0,\*ptr,\*ret,\*str;int len=7,i=0,j;

cJSON \*child=item->child;

int numentries=0,fail=0;

size\_t tmplen=0;

/\* Count the number of entries. \*/

while (child) numentries++,child=child->next;

/\* Explicitly handle empty object case \*/

if (!numentries)

{

if (p) out=ensure(p,fmt?depth+4:3);

else out=(char\*)cJSON\_malloc(fmt?depth+4:3);

if (!out) return 0;

ptr=out;\*ptr++='{';

if (fmt) {\*ptr++='\n';for (i=0;i<depth-1;i++) \*ptr++='\t';}

\*ptr++='}';\*ptr++=0;

return out;

}

if (p)

{

/\* Compose the output: \*/

i=p->offset;

len=fmt?2:1; ptr=ensure(p,len+1); if (!ptr) return 0;

\*ptr++='{'; if (fmt) \*ptr++='\n'; \*ptr=0; p->offset+=len;

child=item->child;depth++;

while (child)

{

if (fmt)

{

ptr=ensure(p,depth); if (!ptr) return 0;

for (j=0;j<depth;j++) \*ptr++='\t';

p->offset+=depth;

}

print\_string\_ptr(child->string,p);

p->offset=update(p);

len=fmt?2:1;

ptr=ensure(p,len); if (!ptr) return 0;

\*ptr++=':';if (fmt) \*ptr++='\t';

p->offset+=len;

print\_value(child,depth,fmt,p);

p->offset=update(p);

len=(fmt?1:0)+(child->next?1:0);

ptr=ensure(p,len+1); if (!ptr) return 0;

if (child->next) \*ptr++=',';

if (fmt) \*ptr++='\n';\*ptr=0;

p->offset+=len;

child=child->next;

}

ptr=ensure(p,fmt?(depth+1):2); if (!ptr) return 0;

if (fmt) for (i=0;i<depth-1;i++) \*ptr++='\t';

\*ptr++='}';\*ptr=0;

out=(p->buffer)+i;

}

else

{

/\* Allocate space for the names and the objects \*/

entries=(char\*\*)cJSON\_malloc(numentries\*sizeof(char\*));

if (!entries) return 0;

names=(char\*\*)cJSON\_malloc(numentries\*sizeof(char\*));

if (!names) {cJSON\_free(entries);return 0;}

memset(entries,0,sizeof(char\*)\*numentries);

memset(names,0,sizeof(char\*)\*numentries);

/\* Collect all the results into our arrays: \*/

child=item->child;depth++;if (fmt) len+=depth;

while (child)

{

names[i]=str=print\_string\_ptr(child->string,0);

entries[i++]=ret=print\_value(child,depth,fmt,0);

if (str && ret) len+=strlen(ret)+strlen(str)+2+(fmt?2+depth:0); else fail=1;

child=child->next;

}

/\* Try to allocate the output string \*/

if (!fail) out=(char\*)cJSON\_malloc(len);

if (!out) fail=1;

/\* Handle failure \*/

if (fail)

{

for (i=0;i<numentries;i++) {if (names[i]) cJSON\_free(names[i]);if (entries[i]) cJSON\_free(entries[i]);}

cJSON\_free(names);cJSON\_free(entries);

return 0;

}

/\* Compose the output: \*/

\*out='{';ptr=out+1;if (fmt)\*ptr++='\n';\*ptr=0;

for (i=0;i<numentries;i++)

{

if (fmt) for (j=0;j<depth;j++) \*ptr++='\t';

tmplen=strlen(names[i]);memcpy(ptr,names[i],tmplen);ptr+=tmplen;

\*ptr++=':';if (fmt) \*ptr++='\t';

strcpy(ptr,entries[i]);ptr+=strlen(entries[i]);

if (i!=numentries-1) \*ptr++=',';

if (fmt) \*ptr++='\n';\*ptr=0;

cJSON\_free(names[i]);cJSON\_free(entries[i]);

}

cJSON\_free(names);cJSON\_free(entries);

if (fmt) for (i=0;i<depth-1;i++) \*ptr++='\t';

\*ptr++='}';\*ptr++=0;

}

return out;

}

其余函数信息如下：

// 返回节点的个数

int cJSON\_GetArraySize(cJSON \*array)

{

cJSON \*c=array->child;

int i=0;

while(c)

i++,c=c->next;

return i;

}

// 返回array中第item个节点的地址

cJSON \*cJSON\_GetArrayItem(cJSON \*array,int item)

{

cJSON \*c=array->child;

while (c && item>0)

item--,c=c->next;

return c;

}

// 返回Object中第item个节点的地址

cJSON \*cJSON\_GetObjectItem(cJSON \*object,const char \*string)

{

cJSON \*c=object->child;

while (c && cJSON\_strcasecmp(c->string,string))

c=c->next;

return c;

}

// 在链表中插入一个新的节点

void cJSON\_InsertItemInArray(cJSON \*array,int which,cJSON \*newitem)

{

cJSON \*c=array->child;

// 找到which位置

while (c && which>0)

c=c->next,which--;

// 添加新的节点到array中

if (!c)

{

cJSON\_AddItemToArray(array,newitem);

return;

}

// 将链表节点进行挂接

newitem->next=c;

newitem->prev=c->prev;

c->prev=newitem;

// 处理arrya的孩子节点

if (c==array->child)

array->child=newitem;

else

newitem->prev->next=newitem;

}

// 替换节点操作，用新的节点替换原有的某一个节点

void cJSON\_ReplaceItemInArray(cJSON \*array,int which,cJSON \*newitem)

{

cJSON \*c=array->child;

// 找到which位置

while (c && which>0)

c=c->next,which--;

if (!c)

return;

// 进行挂接

newitem->next=c->next;

newitem->prev=c->prev;

// 处理NULL情况

if (newitem->next)

newitem->next->prev=newitem;

// 处理孩子节点

if (c==array->child)

array->child=newitem;

else

newitem->prev->next=newitem;

c->next=c->prev=0;

// 删除替换的节点

cJSON\_Delete(c);

}

// 替换节点操作

// 用原有节点替换现有节点

void cJSON\_ReplaceItemInObject(cJSON \*object,const char \*string,cJSON \*newitem)

{

int i=0;

cJSON \*c=object->child;

while(c && cJSON\_strcasecmp(c->string,string))

i++,c=c->next;

if(c)

{

newitem->string=cJSON\_strdup(string);

cJSON\_ReplaceItemInArray(object,i,newitem);

}

}

/\* Create basic types: \*/

// 创建基本类型函数

cJSON \*cJSON\_CreateNull(void)

{

cJSON \*item=cJSON\_New\_Item();

if(item)

item->type=cJSON\_NULL;

return item;

}

cJSON \*cJSON\_CreateTrue(void)

{

cJSON \*item=cJSON\_New\_Item();

if(item)

item->type=cJSON\_True;

return item;

}

cJSON \*cJSON\_CreateFalse(void)

{

cJSON \*item=cJSON\_New\_Item();

if(item)

item->type=cJSON\_False;

return item;

}

cJSON \*cJSON\_CreateBool(int b)

{

cJSON \*item=cJSON\_New\_Item();

if(item)

item->type=b?cJSON\_True:cJSON\_False;

return item;

}

cJSON \*cJSON\_CreateNumber(double num)

{

cJSON \*item=cJSON\_New\_Item();

if(item)

{

item->type=cJSON\_Number;

item->valuedouble=num;

item->valueint=(int)num;

}

return item;

}

cJSON \*cJSON\_CreateString(const char \*string)

{

cJSON \*item=cJSON\_New\_Item();

if(item)

{

item->type=cJSON\_String;

item->valuestring=cJSON\_strdup(string);

}

return item;

}

cJSON \*cJSON\_CreateArray(void)

{

cJSON \*item=cJSON\_New\_Item();

if(item)

item->type=cJSON\_Array;

return item;

}

cJSON \*cJSON\_CreateObject(void)

{

cJSON \*item=cJSON\_New\_Item();

if(item)

item->type=cJSON\_Object;

return item;

}

/\* Create Arrays: \*/

// 创建array

cJSON \*cJSON\_CreateIntArray(const int \*numbers,int count)

{

int i;

cJSON \*n=0,\*p=0,\*a=cJSON\_CreateArray();

for(i=0;a && i<count;i++)

{

n=cJSON\_CreateNumber(numbers[i]);

if(!i)

a->child=n;

else

suffix\_object(p,n);

p=n;

}

return a;

}

cJSON \*cJSON\_CreateFloatArray(const float \*numbers,int count)

{

int i;

cJSON \*n=0,\*p=0,\*a=cJSON\_CreateArray();

for(i=0;a && i<count;i++)

{

n=cJSON\_CreateNumber(numbers[i]);

if(!i)

a->child=n;

else

suffix\_object(p,n);

p=n;

}

return a;

}

cJSON \*cJSON\_CreateDoubleArray(const double \*numbers,int count)

{

int i;

cJSON \*n=0,\*p=0,\*a=cJSON\_CreateArray();

for(i=0;a && i<count;i++)

{

n=cJSON\_CreateNumber(numbers[i]);

if(!i)

a->child=n;

else

suffix\_object(p,n);

p=n;

}

return a;

}

cJSON \*cJSON\_CreateStringArray(const char \*\*strings,int count)

{

int i;

cJSON \*n=0,\*p=0,\*a=cJSON\_CreateArray();

for(i=0;a && i<count;i++)

{

n=cJSON\_CreateString(strings[i]);

if(!i)

a->child=n;

else

suffix\_object(p,n);

p=n;

}

return a;

}

/\* Duplication \*/

// 拷贝副本操作

cJSON \*cJSON\_Duplicate(cJSON \*item,int recurse)

{

cJSON \*newitem,\*cptr,\*nptr=0,\*newchild;

/\* Bail on bad ptr \*/

if (!item)

return 0;

/\* Create new item \*/

newitem=cJSON\_New\_Item();

if (!newitem)

return 0;

/\* Copy over all vars \*/

newitem->type=item->type&(~cJSON\_IsReference),newitem->valueint=item->valueint,newitem->valuedouble=item->valuedouble;

if (item->valuestring)

{

newitem->valuestring=cJSON\_strdup(item->valuestring);

if (!newitem->valuestring)

{

cJSON\_Delete(newitem);

return 0;

}

}

if (item->string)

{

newitem->string=cJSON\_strdup(item->string);

if (!newitem->string)

{

cJSON\_Delete(newitem);

return 0;

}

}

/\* If non-recursive, then we're done! \*/

if (!recurse)

return newitem;

/\* Walk the ->next chain for the child. \*/

cptr=item->child;

while (cptr)

{

newchild=cJSON\_Duplicate(cptr,1); /\* Duplicate (with recurse) each item in the ->next chain \*/

if (!newchild)

{

cJSON\_Delete(newitem);

return 0;

}

if (nptr)

{

nptr->next=newchild,newchild->prev=nptr;

nptr=newchild;

} /\* If newitem->child already set, then crosswire ->prev and ->next and move on \*/

else

{

newitem->child=newchild;

nptr=newchild;

} /\* Set newitem->child and move to it \*/

cptr=cptr->next;

}

return newitem;

}

void cJSON\_Minify(char \*json)

{

char \*into=json;

while (\*json)

{

if (\*json==' ') json++;

else if (\*json=='\t') json++; /\* Whitespace characters. \*/

else if (\*json=='\r') json++;

else if (\*json=='\n') json++;

else if (\*json=='/' && json[1]=='/') while (\*json && \*json!='\n') json++; /\* double-slash comments, to end of line. \*/

else if (\*json=='/' && json[1]=='\*') {while (\*json && !(\*json=='\*' && json[1]=='/')) json++;json+=2;} /\* multiline comments. \*/

else if (\*json=='\"'){\*into++=\*json++;while (\*json && \*json!='\"'){if (\*json=='\\') \*into++=\*json++;\*into++=\*json++;}\*into++=\*json++;} /\* string literals, which are \" sensitive. \*/

else \*into++=\*json++; /\* All other characters. \*/

}

\*into=0; /\* and null-terminate. \*/

}

* **参考文献**

<http://www.0xffffff.org/2014/02/10/29-cjson-analyse/>

<http://github.tiankonguse.com/blog/2014/12/18/cjson-source.html>

<http://www.codexiu.cn/javascript/blog/21402/>

<http://www.json.org/>

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/yzhang6\_10/article/details/51615089