JSON c 语言开发指南

版本	修改记录	日期	修订人

1. 引言

本文档是基于 json-c 库对数据交换进行开发所编写的开发指南，及详细解释 json-c 库中常用 api。 适用于开发人员使用 c 语言对 json 的编程。

（注： 此文档 json-c 库版本为 0.8——json-c-0.8）

2. JSON 简介

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。易于人阅读和编写。同时也易于机器解析和生成。它基于 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（包括 C, C++, C#, Java, JavaScript, Perl, Python 等)。这些特性使 JSON 成为理想的数据交换语言。

跟 XML 相比，JSON 的优势在于格式简洁短小，特别是在处理大量复杂数据的时候，这个优势便显得非常突出。从各浏览器的支持来看，JSON 解决了因不同浏览器对 XML DOM 解析方式不同而引起的问题。

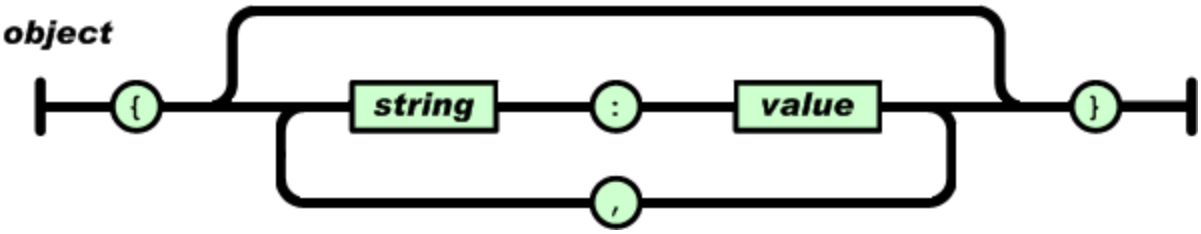
2.1 JSON 建构于两种结构：

- “名称/值”对的集合（A collection of name/value pairs）。不同的语言中，它被理解为*对象（object）*，纪录（record），结构（struct），字典（dictionary），哈希表(hash table)，有键列表(keyed 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）。

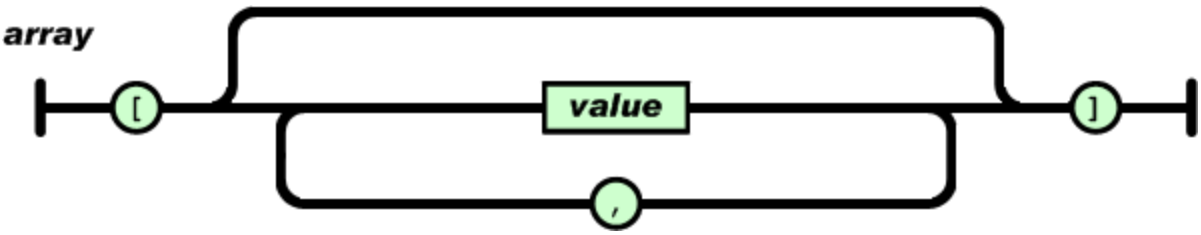
这些都是常见的数据结构。事实上大部分现代计算机语言都以某种形式支持它们。这使得一种数据格式在同样基于这些结构的编程语言之间交换成为可能。

2.2 JSON 具有以下这些形式：

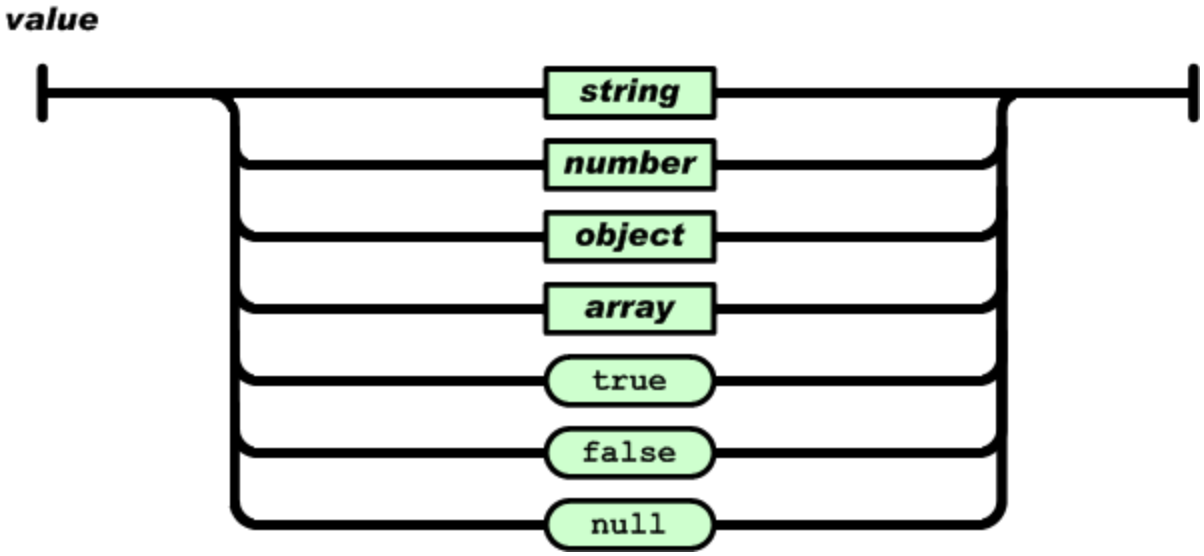
对象是一个无序的“‘名称/值’对”集合。一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）；“‘名称/值’ 对”之间使用“,”（逗号）分隔。



数组是值（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）分隔。

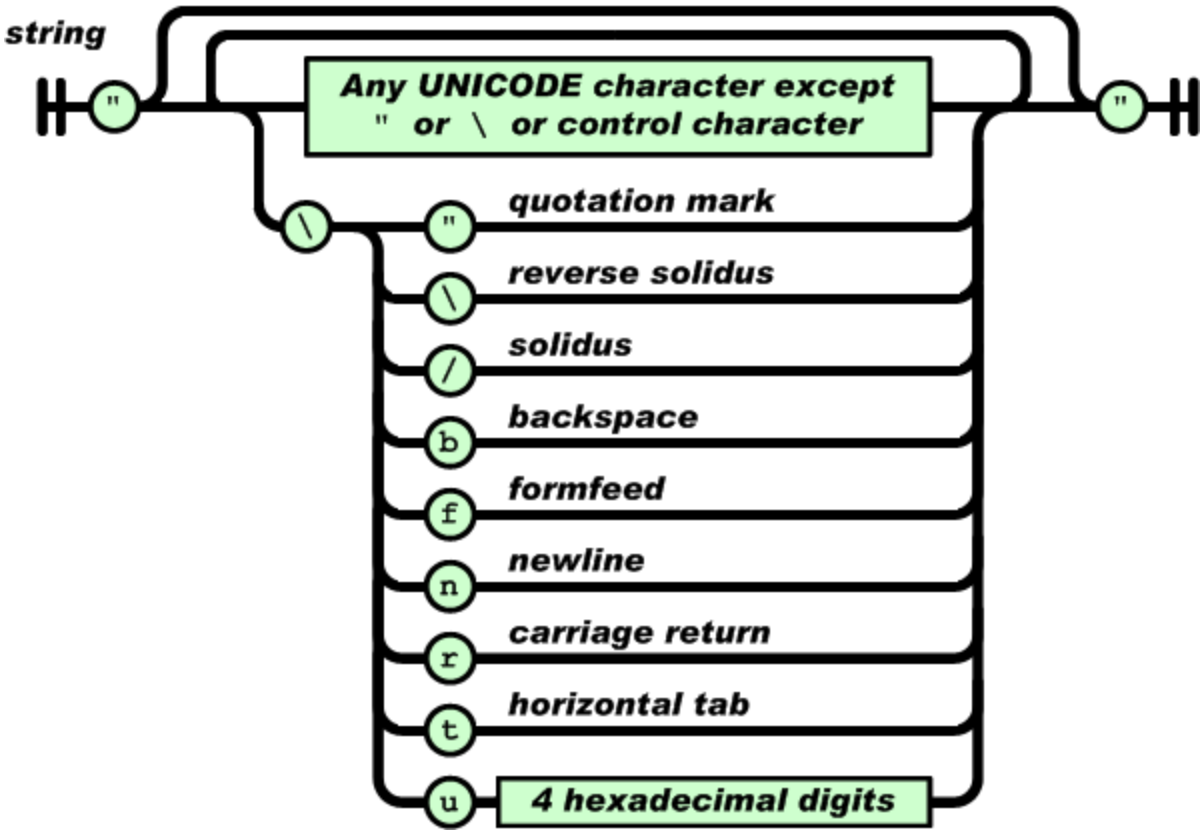


值（*value*）可以是双引号括起来的字符串（*string*）、数值(number)、true、false、 null、对象（object）或者数组（array）。这些结构可以嵌套。

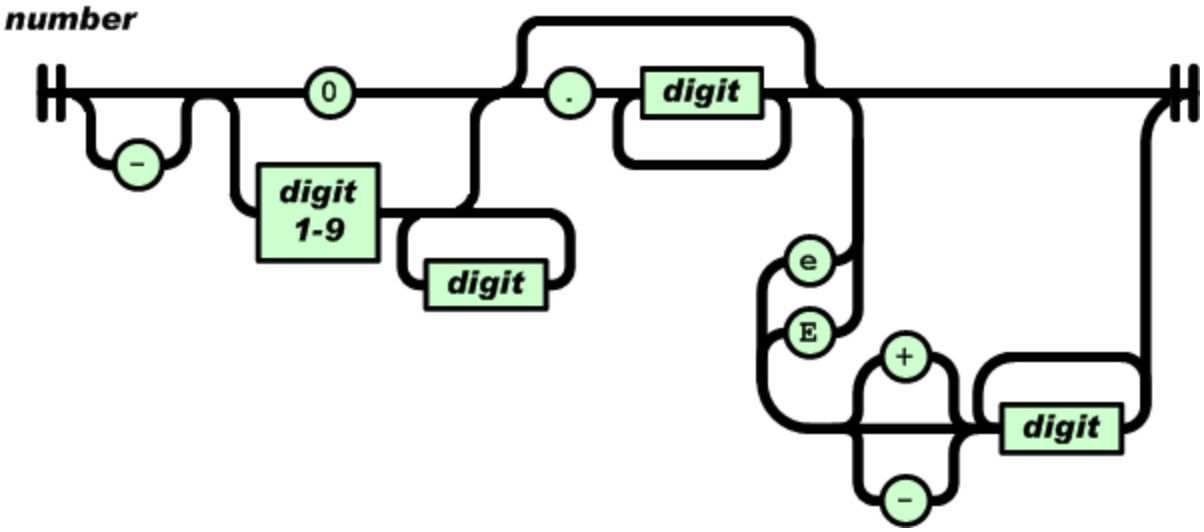


字符串（*string*）是由双引号包围的任意数量 Unicode 字符的集合，使用反斜线转义。一个字符（character）即一个单独的字符串（character string）。

字符串（*string*）与 C 或者 Java 的字符串非常相似。



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。



3. JSON 库函数说明

3.1 JSON 对象的生成

Json 对象的类型：

json_type_object, “名称/值”对的集合

Json 对象的值类型

json_type_boolean,
json_type_double,
json_type_int,
json_type_array, “值”的集合
json_type_string

struct json_object * json_object_new_object();

说明：

创建个空的 json_type_object 类型 JSON 对象。

struct json_object* json_object_new_boolean(boolean b);

说明：

创建个 json_type_boolean 值类型 json 对象

boolean json_object_get_boolean(struct json_object *obj);

说明：

从 json 对象中 boolean 值类型得到 boolean 值

同样：

struct json_object* json_object_new_int(int i)
int json_object_get_int(struct json_object *this)
struct json_object* json_object_new_double(double d)
double json_object_get_double(struct json_object *this)
struct json_object* json_object_new_string(char *s)
char* json_object_get_string(struct json_object *this)

struct json_object * json_object_new_array();

说明：

创建个空的 json_type_array 类型 JSON 数组值对象。

struct json_object * json_tokener_parse(char *str);

说明：

由 str 里的 JSON 字符串生成 JSON 对象,str 是 son_object_to_json_string() 生成的。

参数：

str - json 字符串

struct json_object * json_object_object_get(struct json_object * json,char *name);

说明：

从 json 中按名字取一个对象。

参数：

json - json 对象
name - json 域名字

3.2 JSON 对象的释放

struct json_object * * json_object_get(struct json_object * this)

说明：

增加对象引用计数。使用 c 库最关心的是内存谁来分配，谁来释放. jsonc 的内存管理方式，是基于引用计数的内存树(链)，如果把一个 struct json_object 对象 a, add 到另一个对象 b 上，就不用显式的释放(json_object_put) a 了，相当于把 a 挂到了 b 的对象树上，释放 b 的时候，就会释放 a. 当 a 即 add 到 b 上，又 add 到对象 c 上时会导致 a 被释放两次(double free)，这时可以增加 a 的引用计数(调用函数 json_object_get(a))，这时如果先释放 b，后释放 c，当释放 b 时，并不会真正的释放 a，而是减少 a 的引用计数为 1，然后释放 c 时，才真正释放 a.

参数：

this – json 对象

Void json_object_put(struct json_object * this)

说明：

减少对象引用次数一次，当减少到 0 就释放（free）资源

参数：

this – json 对象

样例片段：

```
my_string = json_object_new_string("\t");
/*输出 my_string=     */ \t(table 键)
printf("my_string=%s\n", json_object_get_string(my_string));
/*转换 json 格式字符串 输出 my_string.to_string()="\t"*/
printf("my_string.to_string()=%s\n", json_object_to_json_string(my_string));
/*释放资源*/
json_object_put(my_string);
```

3.3 JSON 对象的操作

Int json_object_is_type(struct json_object * this, enum json_type type)

说明：

检查 json_object 是 json 的某个类型

参数：

this: json_object 实例

type: json_type_boolean,json_type_double, json_type_int, json_type_object, json_type_array, json_type_string

enum json_type json_object_get_type(struct json_object * this)

说明：

得到 json_object 的类型。

参数：

this – json 对象

char * json_object_to_json_string(struct json_object * this)

说明：

将 json_object 内容转换 json 格式字符串，其中可能含有转义符。

参数：

this – json 对象

返回值：

Json 格式字符串

void json_object_object_add(struct json_object* obj, char *key, struct json_object *val);

说明：

添加个对象域到 json 对象中

参数：

Obj – json 对象

key – 域名字

val – json 值对象

void json_object_object_del(struct json_object* obj, char *key);

说明：

删除 key 值 json 对象

参数：

ob j – json 对象

key – 域名字

int json_object_array_length(struct json_object *obj);

说明：

得到 json 对象数组的长度。

参数：

ob j – json 数组值对象

extern int json_object_array_add(struct json_object *obj,
struct json_object *val);

说明：

添加一元素在 json 对象数组末端

参数：

ob j – json 数组值对象

val – json 值对象

*

int json_object_array_put_idx(struct json_object *obj, int idx,
                              struct json_object *val);

说明：

在指定的 json 对象数组下标插入或替换一个 json 对象元素。

参数：

ob j – json 数组值对象

val – json 值对象

idx – 数组下标

struct json_object * json_object_array_get_idx(struct json_object * json_array, int i);

说明：

从数组中，按下标取 JSON 值对象。

参数：

json_array – json 数组类型对象

i – 数组下标位置

定义宏 json_object_object_foreach(obj,key,val)

说明：

遍历 json 对象的 key 和值 （key, val 默认参数不变）

样例片段：

```
/*创建个空 json 对象值数组类型*/
my_array = json_object_new_array();
/*添加 json 值类型到数组中*/
json_object_array_add(my_array, json_object_new_int(1));
json_object_array_add(my_array, json_object_new_int(2));
json_object_array_add(my_array, json_object_new_int(3));
json_object_array_put_idx(my_array, 4, json_object_new_int(5));
printf("my_array=\n");
for(i=0; i < json_object_array_length(my_array); i++) {
   struct json_object *obj = json_object_array_get_idx(my_array, i);
   printf("\t[%d]=%s\n", i, json_object_to_json_string(obj));
}
printf("my_array.to_string()=%s\n", json_object_to_json_string(my_array));

my_object = json_object_new_object();
/*添加 json 名称和值到 json 对象集合中*/
json_object_object_add(my_object, "abc", json_object_new_int(12));
json_object_object_add(my_object, "foo", json_object_new_string("bar"));
json_object_object_add(my_object, "bool0", json_object_new_boolean(0));
json_object_object_add(my_object, "bool1", json_object_new_boolean(1));
```

```
    json_object_object_add(my_object, "baz", json_object_new_string("bang"));
    /*同样的 key  添加会替换掉*/
    json_object_object_add(my_object, "baz", json_object_new_string("fark"));
    json_object_object_del(my_object, "baz");
    /*添加数组集合到 json 对象中*/
    json_object_object_add(my_object, "arr", my_array);

    printf("my_object=\n");
    /*遍历 json 对象集合*/
    json_object_object_foreach(my_object, key, val) {
        printf("\t%s: %s\n", key, json_object_to_json_string(val));
    }
    json_object_put(my_object);
```

4. JSON 实例开发

4.1 样例 1

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>

#include "json.h"

int main(int argc, char **argv)
{
    struct json_tokener *tok;
    struct json_object *my_string, *my_int, *my_object, *my_array;
    struct json_object *new_obj;
    int i;

    my_string = json_object_new_string("\t");
    /*输出  my_string=     */
    printf("my_string=%s\n", json_object_get_string(my_string));
    /*转换 json 格式字符串  输出 my_string.to_string()="\t"*/
    printf("my_string.to_string()=%s\n", json_object_to_json_string(my_string));
    /*释放资源*/
    json_object_put(my_string);

    my_string = json_object_new_string("\\");
    printf("my_string=%s\n", json_object_get_string(my_string));
```

```
  printf("my_string.to_string()=%s\n", json_object_to_json_string(my_string));
  json_object_put(my_string);

  my_string = json_object_new_string("foo");
  printf("my_string=%s\n", json_object_get_string(my_string));
  printf("my_string.to_string()=%s\n", json_object_to_json_string(my_string));

  my_int = json_object_new_int(9);
  printf("my_int=%d\n", json_object_get_int(my_int));
  printf("my_int.to_string()=%s\n", json_object_to_json_string(my_int));

  /*创建个空 json 对象值数组类型*/
  my_array = json_object_new_array();
  /*添加 json 值类型到数组中*/
  json_object_array_add(my_array, json_object_new_int(1));
  json_object_array_add(my_array, json_object_new_int(2));
  json_object_array_add(my_array, json_object_new_int(3));
  json_object_array_put_idx(my_array, 4, json_object_new_int(5));
  printf("my_array=\n");
  for(i=0; i < json_object_array_length(my_array); i++) {
     struct json_object *obj = json_object_array_get_idx(my_array, i);
     printf("\t[%d]=%s\n", i, json_object_to_json_string(obj));
  }
  printf("my_array.to_string()=%s\n", json_object_to_json_string(my_array));

  my_object = json_object_new_object();
  /*添加 json 名称和值到 json 对象集合中*/
  json_object_object_add(my_object, "abc", json_object_new_int(12));
  json_object_object_add(my_object, "foo", json_object_new_string("bar"));
  json_object_object_add(my_object, "bool0", json_object_new_boolean(0));
  json_object_object_add(my_object, "bool1", json_object_new_boolean(1));
  json_object_object_add(my_object, "baz", json_object_new_string("bang"));
  /*同样的 key  添加会替换掉*/
  json_object_object_add(my_object, "baz", json_object_new_string("fark"));
  json_object_object_del(my_object, "baz");

  printf("my_object=\n");
  /*遍历 json 对象集合*/
  json_object_object_foreach(my_object, key, val) {
     printf("\t%s: %s\n", key, json_object_to_json_string(val));
  }
  printf("my_object.to_string()=%s\n", json_object_to_json_string(my_object));

/*对些不规则的串做了些解析测试*/
```

```
new_obj = json_tokener_parse("\"\003\"");
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
json_object_put(new_obj);

new_obj = json_tokener_parse("/* hello */\"foo\"");
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
json_object_put(new_obj);

new_obj = json_tokener_parse("// hello\n\"foo\"");
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
json_object_put(new_obj);

new_obj = json_tokener_parse("\"\\u0041\\u0042\\u0043\"");
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
json_object_put(new_obj);

new_obj = json_tokener_parse("null");
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
json_object_put(new_obj);

new_obj = json_tokener_parse("True");
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
json_object_put(new_obj);

new_obj = json_tokener_parse("12");
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
json_object_put(new_obj);


new_obj = json_tokener_parse("12.3");
/*得到 json double 类型
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
json_object_put(new_obj);

new_obj = json_tokener_parse("[\"\\n\"]");
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
json_object_put(new_obj);

new_obj = json_tokener_parse("[\"\\nabc\\n\"]");
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
json_object_put(new_obj);

new_obj = json_tokener_parse("[null]");
printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
```

```
    json_object_put(new_obj);

    new_obj = json_tokener_parse("[]");
    printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
    json_object_put(new_obj);

    new_obj = json_tokener_parse("[false]");
    printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
    json_object_put(new_obj);

    new_obj = json_tokener_parse("[\"abc\",null,\"def\",12]");
    printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
    json_object_put(new_obj);

    new_obj = json_tokener_parse("{}");
    printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
    json_object_put(new_obj);

    new_obj = json_tokener_parse("{ \"foo\": \"bar\" }");
    printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
    json_object_put(new_obj);

    new_obj = json_tokener_parse("{ \"foo\": \"bar\", \"baz\": null, \"bool0\": true }");
    printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
    json_object_put(new_obj);

    new_obj = json_tokener_parse("{ \"foo\": [null, \"foo\"] }");
    printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
    json_object_put(new_obj);

    new_obj = json_tokener_parse("{ \"abc\": 12, \"foo\": \"bar\", \"bool0\": false, \"bool1\": true,
\"arr\": [ 1, 2, 3, null, 5 ] }");
    printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
    json_object_put(new_obj);

    new_obj = json_tokener_parse("{ foo }");
    if(is_error(new_obj)) printf("got error as expected\n");

    new_obj = json_tokener_parse("foo");
    if(is_error(new_obj)) printf("got error as expected\n");

    new_obj = json_tokener_parse("{ \"foo");
    if(is_error(new_obj)) printf("got error as expected\n");
```

```
    /* test incremental parsing */
    tok = json_tokener_new();
    new_obj = json_tokener_parse_ex(tok, "{ \"foo", 6);
    if(is_error(new_obj)) printf("got error as expected\n");
    new_obj = json_tokener_parse_ex(tok, "\": {\"bar", 8);
    if(is_error(new_obj)) printf("got error as expected\n");
    new_obj = json_tokener_parse_ex(tok, "\":13}}", 6);
    printf("new_obj.to_string()=%s\n", json_object_to_json_string(new_obj));
    json_object_put(new_obj);
    json_tokener_free(tok);

    json_object_put(my_string);
    json_object_put(my_int);
    json_object_put(my_object);
    json_object_put(my_array);
/*如果前面没有添加到对象中，  必须显示释放，
  如果添加到对象中，已经释放对象，则无需调用，  在这务必小心，否则很容易内存泄漏*/

    return 0;
}
```

输出结果：

```
my_string=
my_string.to_string()="\t"
my_string=\
my_string.to_string()="\\"
my_string=foo
my_string.to_string()="foo"
my_int=9
my_int.to_string()=9
my_array=
        [0]=1
        [1]=2
        [2]=3
        [3]=null
        [4]=5
my_array.to_string()=[ 1, 2, 3, null, 5 ]
my_object=
        abc: 12
        foo: "bar"
        bool0: false
        bool1: true
my_object.to_string()={ "abc": 12, "foo": "bar", "bool0": false, "bool1": true }
```

```
new_obj.to_string()="\u0003"
new_obj.to_string()="foo"
new_obj.to_string()="foo"
new_obj.to_string()="ABC"
new_obj.to_string()=null
new_obj.to_string()=true
new_obj.to_string()=12
new_obj.to_string()=12.300000
new_obj.to_string()=[ "\n" ]
new_obj.to_string()=[ "\nabc\n" ]
new_obj.to_string()=[ null ]
new_obj.to_string()=[ ]
new_obj.to_string()=[ false ]
new_obj.to_string()=[ "abc", null, "def", 12 ]
new_obj.to_string()={ }
new_obj.to_string()={ "foo": "bar" }
new_obj.to_string()={ "foo": "bar", "baz": null, "bool0": true }
new_obj.to_string()={ "foo": [ null, "foo" ] }
new_obj.to_string()={ "abc": 12, "foo": "bar", "bool0": false, "bool1": true, "arr": [ 1, 2, 3, null,
5 ] }
got error as expected
got error as expected
got error as expected
new_obj.to_string()={ "foo": { "bar": 13 } }
```

备注

Json-c-0.8 版本对 0.7 版本做的更新：

* 添加 va_end 给指针至 NULL 增加程序健壮性
* 添加宏使得能够编译出调试代码
* 解决个 bug 在指数中使用大写字母 E
* 添加 stddef.h 头文件
* 允许编译 json-c 使用-Werror