

学习目标

xml

- xml基础语法和规范
- C程序中如何使用xml开源库
- 借助开源库,在C程序中生成xml文件
- 已知一个xml文件,如何借助开源库解析xml文件数据

Json

- json的基础语法和规范
- C程序中如何使用json开源库 - cJSON
- 使用cjson生成json文件
- 已知一个json文件,使用cjson库解析文件数据

XML

minixml官网地址

<http://www.msweet.org/projects.php/Mini-XML>

其他解析xml开源库: tinyxml pugixml

- 包含头文件: mxml.h
- 编译的时候需要添加动态库: libmxml.so
 - -lmxml
 - /usr/local/lib

1. minixml安装:

- ./configure --enable-threads=no && make
- sudo make install

2. 新目录需要一个文件头 - 标准

<?xml version="1.0" encoding="utf-8"?>

- version不可以省略
- encoding可以省略

3. 使用注意事项

- 必须有一个根元素(节点) -- (只有一个)
- xml标签对大小写敏感
- 标签大多成对使用, 有开始, 有结束
 - <date></date>
 - <time></time>

4. 标签中可以添加属性

- <node date="17/11/2017">
- 属性值必须加引号.

5. 标签注释

- <!-- 这是注释 -->

6. 开源库minixml的使用

- 跟标签的对应的节点, 父亲节点是: 文件头节点

○ 生成xml文件

- 创建一个新的xml文件

mxml_node_t *mxmlNewXML(const char *version);

- 返回新创建的xml文件节点.

□ 默认的文件编码为utf8

- 删除节点的内存

```
void mxmlDelete(mxml_node_t *node);
```

- 添加一个新的节点

```
mxml_node_t *mxmlNewElement(  
    mxml_node_t *parent, // 父节点  
    const char *name      // 新节点标签名  
);
```

- 设置节点的属性名和属性值

```
void mxmlElementSetAttr(  
    mxml_node_t *node, // 被设置属性的节点  
    const char *name,   // 节点的属性名  
    const char *value   // 属性值  
);
```

- 创建节点的文本内容

```
mxml_node_t *mxmlNewText(  
    mxml_node_t *parent, // 节点地址  
    int whitespace,      // 是否有空白 0  
    const char *string    // 文本内容  
);
```

- 保存节点到xml文件

```
int mxmlSaveFile(  
    mxml_node_t *node, // 根节点  
    FILE *fp,          // 文件指针  
    mxml_save_cb_t cb   // 默认MXML_NO_CALLBACK  
);
```

○ 解析xml文件

- 从文件加载xml到内存

```
mxml_node_t *mxmlLoadFile(  
    mxml_node_t *top, // 一般为NULL  
    FILE *fp,         // 文件指针  
    mxml_type_t (*cb)(mxml_node_t *) // 默认MXML_NO_CALLBACK  
);
```

- 获取节点的属性

```
const char *mxmlElementGetAttr(  

```

```

    mxml_node_t *node, // 带属性的节点的地址
    const char *name    // 属性名
);

```

- 获取指定节点的文本内容

```

const char *mxmlGetText(
    mxml_node_t *node, // 节点的地址
    int *whitespace    // 是否有空格
);

```
- 跳转到下一个节点

```

mxml_node_t *mxmlWalkNext(
    mxml_node_t *node, // 当前节点
    mxml_node_t *top,  // 根节点
    int descend
);

```

 - descend: 搜索的规则
 - ◆ MXML_NO_DESCEND: 查看同层级
 - ◆ MXML_DESCEND_FIRST: 查看下一层级的第一个
 - ◆ MXML_DESCEND: 一直向下搜索
- 查找节点

```

mxml_node_t *mxmlFindElement(
    mxml_node_t *node,    // 当前节点
    mxml_node_t *top,    // 根节点
    const char *name,     // 查找的标签名
    const char *attr,     // 查找的标签的属性
                        没有属性传NULL
    const char *value,    // 查找的标签的属性值
    int descend           // 同上
);

```

JSON

1. cJSON的使用

- 压缩包解压缩，直接使用里边的cJSON.c和cJSON.h即可
- 链接时还需要加上-lm 表示链接math库

2. json的格式

- json数组
 - o char array[23] = "slkjflajslfd"; - c
 - o 中括号[整形, 字符串, 布尔类型, json数组, json对象]
 - o [123, 123.2, true, false, [12, 34, 56, "hello, world"]]
- json对象
 - o {}中是一些键值对
 - {
 "name":"zhang3",
 "name2":"li4"
 }
 - key值: 必须是 字符串, 不重复
 - value值: json对象, json数组, 布尔, 整形, 字符串
- json数组+json对象

```
{
  "name":"zhang3",
  "name2":"li4",
  "张三":{
    "别名":"老王",
    "性别":"男",
    "年龄":34,
    "孩子":["小红", "小绿", "小黑"]
  }
}
```

3. C语言json开源解析库 - cJSON

- 生成json文件
 - o 创建一个json对象

`cJSON *cJSON_CreateObject(void);`

- 往json对象中添加数据成员

`void cJSON_AddItemToObject(`

`cJSON *object, // json对象`

`const char *string, // key值`

`cJSON *item // value值 (int, string, array, obj)`

`);`

需要将不同类型的数据转为cJSON指针类型

下面的函数就是转换函数

- 创建一个整型值

`cJSON *cJSON_CreateNumber(double num);`

- 创建一个字符串

`cJSON *cJSON_CreateString(const char *string);`

- 创建一个json数组

`cJSON *cJSON_CreateArray(void); -- 空数组`

- 创建默认有count个整形值的json数组

`cJSON *cJSON_CreateIntArray(const int *numbers, int count);`

▪ `int array[] = {8,3,4,5,6};`

▪ `cJSON_CreateIntArray(array, 5);`

- 往json数组中添加数据成员

`void cJSON_AddItemToArray(cJSON *array, cJSON *item);`

- 释放JSON结构指针

`void cJSON_Delete(cJSON *c)`

- 将JSON结构转化为字符串

`char *cJSON_Print(cJSON *item);`

▪ 返回值需要使用free释放

▪ `FILE* fp = fopen();`

▪ `fwrite();`

▪ `fclose();`

需要把json格式文件转为字符串,然后打开文件,写入文件,再关闭文件描述符.

- 解析json文件

- 将字符串解析为JSON结构

`cJSON *cJSON_Parse(const char *value);`

▪ 返回值需要使用cJSON_Delete释放

- 根据键值查找json节点

```
cJSON *cJSON_GetObjectItem(  
    cJSON *object,    // 当前json对象  
    const char *string // key值  
);
```

- 获取json数组中元素的个数

```
int cJSON_GetArraySize(cJSON *array);
```

- 根据数组下标找到对应的数组元素

```
cJSON *cJSON_GetArrayItem(cJSON *array, int index);
```

- 判断是否有可以值对应的键值对

```
int cJSON_HasObjectItem(cJSON *object, const char *string)
```

--- cJSON 结构体

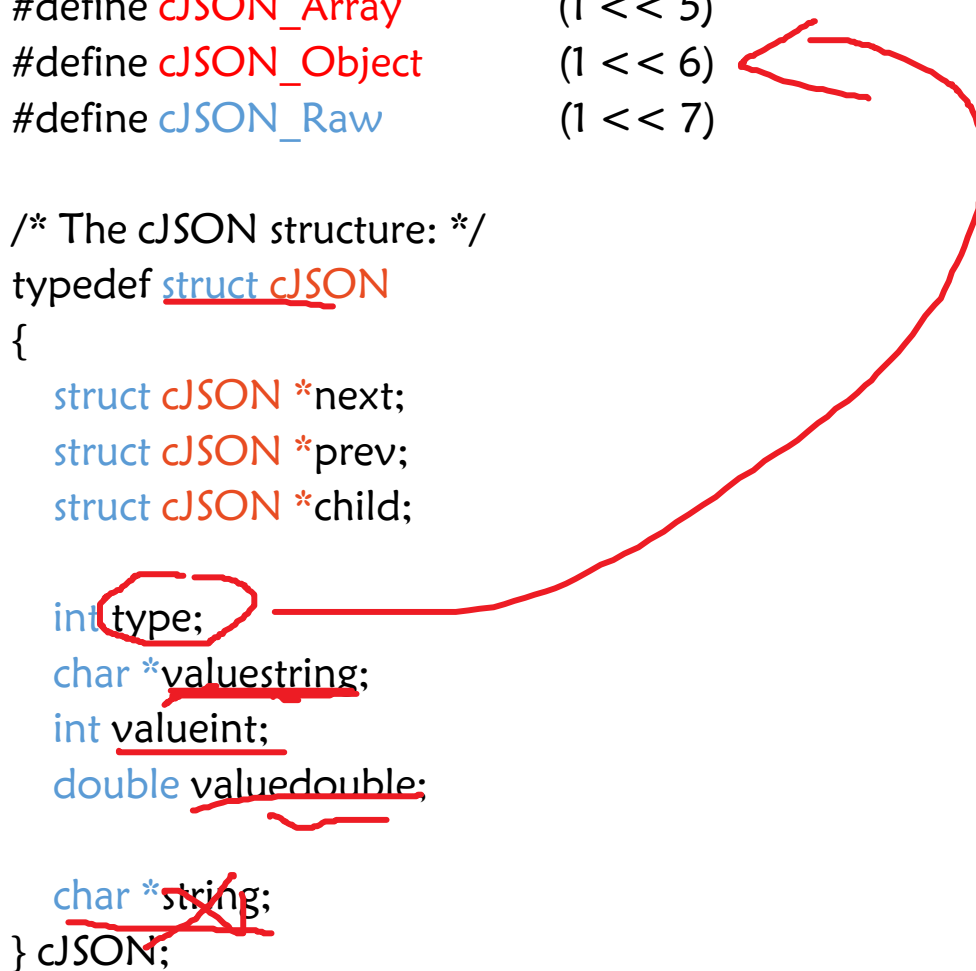
```
#define cJSON_Invalid      (0)
#define cJSON_False       (1 << 0)
#define cJSON_True        (1 << 1)
#define cJSON_NULL        (1 << 2)
#define cJSON_Number      (1 << 3)
#define cJSON_String      (1 << 4)
#define cJSON_Array       (1 << 5)
#define cJSON_Object      (1 << 6)
#define cJSON_Raw         (1 << 7)
```

/* The cJSON structure: */

```
typedef struct cJSON
{
    struct cJSON *next;
    struct cJSON *prev;
    struct cJSON *child;

    int type;
    char *valuestring;
    int valueint;
    double valuedouble;

    char *string;
} cJSON;
```



---.h

typedef int aa;

----.c

aa b;

QT中的json类

2017年6月10日 16:26

QJsonDocument - json文档

QJsonArray - 数组

QJsonObject - 对象

QJsonValue - 对数据的封装