hardccyy的专栏

个人资料

令狐不冲

访问：3193次
积分：111分
排名：千里之外

原创：3篇 转载：26篇
译文：1篇 评论：0条

文章搜索

文章分类

C语言 (24)
指针 (5)
sin (1)
找不到库 (1)
java (1)
linux (10)
so (0)
so库 (0)
libevent (1)
socket (1)
ucarp (1)
时间函数 (1)
vmware (1)
nfs (1)

文章存档

2013年09月 (4)
2013年08月 (1)
2013年07月 (3)
2013年06月 (7)
2013年05月 (15)

阅读排行

JNI_Onload 的用法 版本 (296)
cJson库的使用 (257)
JSON格式解析和libjson (256)
linux下内存泄露检测工具 (188)
cJson 创建 读取 (154)
cjson数组如何解析 (136)

JSON格式解析和libjson使用简介（cJson格式）

分类： C语言 2013-05-22 16:15 256人阅读 评论(0) 收藏 举报

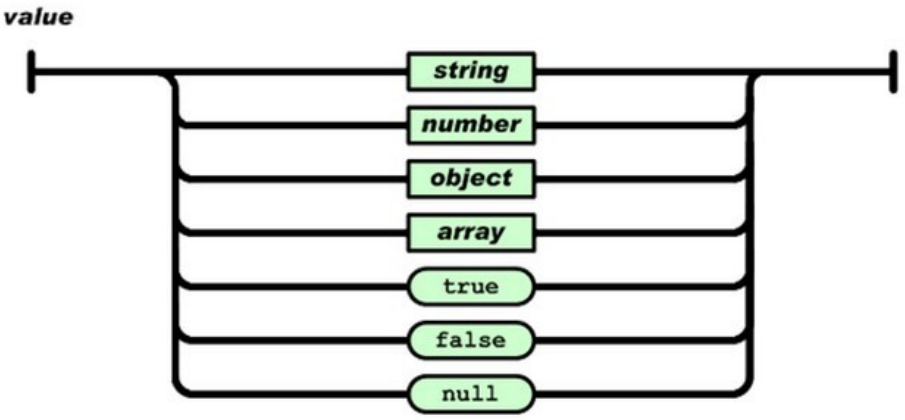
JSON C Linux cJson

Rss Reader实例开发中，进行网络数据交换时主要使用到了两种数据格式：JSON与XML。本文主要介绍JSON格式的简介念及JSON在Rss Reader中的应用。

JSON格式解析和libjson使用简介

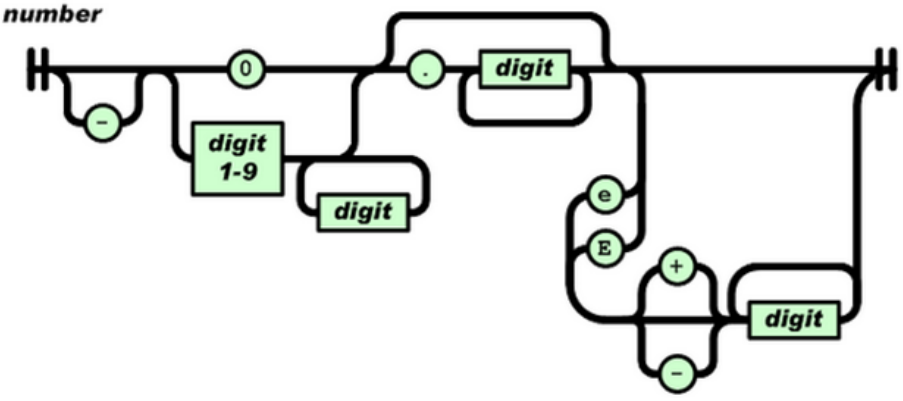
JSON简介：

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，可以把JSON的结构理解成无序的、可嵌套的key-val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键值对之间是用一个“,”（逗号开，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，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的取值集为：Number、Boolean(true false)、null、String、Object及Array，如图一：



（图一）

1、Number：数值，包括整形数与浮点数，如：123、0.83、-2.7e10。其结构如图二：



（图二）

Linux下Eclipse编译时， (132)
Linux下NFS(网络文件系 (129)
libmysqlclient.so.15: cai (123)
使用 libevent 和 libev 提i (119)

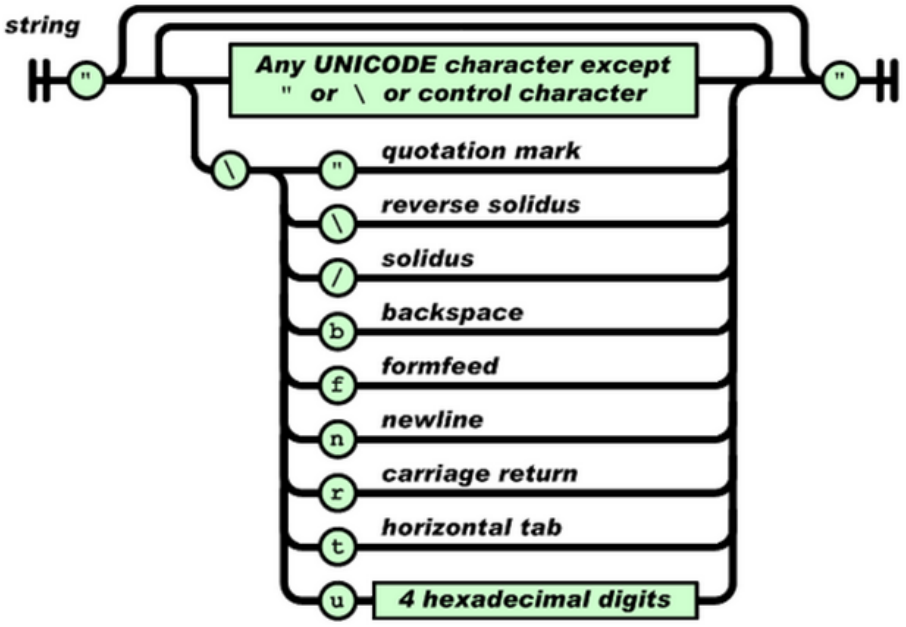
评论排行

指向指针的指针 (0)
Linux Top 命令解析 (0)
共享内存---shmget shm (0)
共享内存函数（shmget (0)
libmysqlclient.so.15: cai (0)
struct sockaddr_in等So((0)
使用 libevent 和 libev 提i (0)
利用ucarp实现虚拟IP故i (0)
fork()函数的用法 (0)
JNI_Onload 的用法 版本 (0)

推荐文章

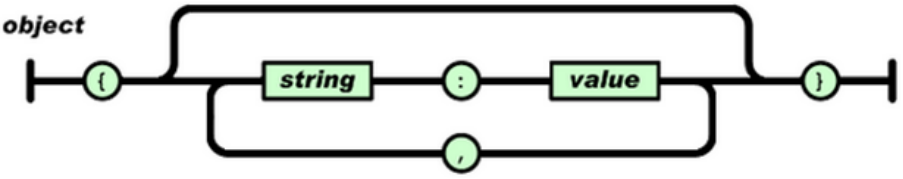
* java网络编程之简单客户端服务器
* 我在ESB上走的弯路
* Java线程(篇外篇)：线程本地变量ThreadLocal
* Android在AsyncHttpClient框架的基础上定制能直接返回对象数组的框架
* 用初次训练的SVM+HOG分类器在负样本原图上检测HardExample
* 在Visual Studio 2008上调试C语言程序

2、String：字符串，是以双引号封闭起来的一串字符，使用反斜杠来转义，如：\\、\n等，JSON中字符串的概念与C/C++或者JAVA语言里的字符串概念差不多，如："abc"。其结构如图三：



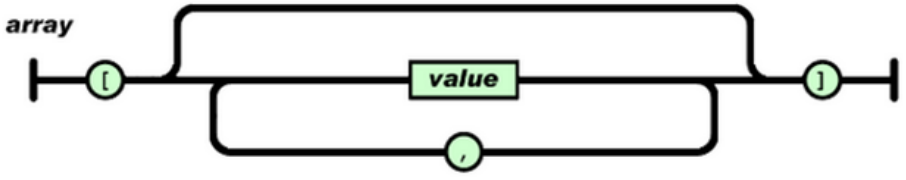
（图三）

3、Object：对象，也可理解成一个结构体，是以一对大括号封闭起来的无序的key-value键值对集合，例如：{name:"Susan", age:27, birthday:{year:1984, month:2, day:11}}；也可以写成：{"name":"Susan", "age":27, "birthday":{"year":1984, "month":2, "day":11}}；其结构如图四：



（图四）

4、Array：数组，JSON的数组是一个以中括号封闭起来的value的集合，即数组内的各个成员的数据类型可以不一样，一点就跟C/JAVA的数组概念不同了。每个value之间是由一个","(逗号)隔开，例如：[123, abc, false, {name:mj}]；其结如图五：



（图五）

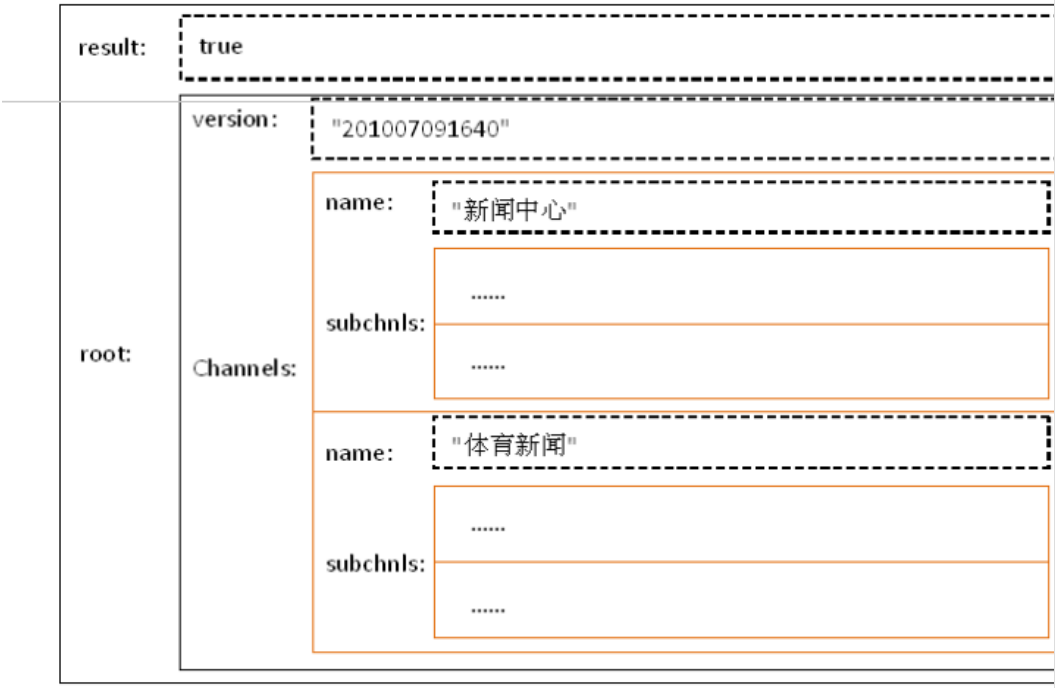
关于JSON的详细说明与教程请自行到网络上搜索，有很多。

下面我们就来动手写一个例子：

```
{
  result:true,

  root:{
    version:"201007091640",
    channels:[
    {
      name:"新闻中心",
      subchnls:[
```

```
        {
          title:"焦点新闻",
          link:"http://news.mtc.sohu.com/news/channel/1/news.rss",
          desc:"家事、国事、天下事"
        },
        {
          title:"新闻频道",
          link:"http://news.mtc.sohu.com/news/channel/2/news.rss",
          desc:"让您实时掌握国际动态"
        },
        {
          title:"军事频道",
          link:"http://news.mtc.sohu.com/news/channel/3/news.rss",
          desc:"军事"
        }
        ]
      },

      {
        name:"体育新闻",
        subchnls:[
        {
          title:"体育要闻汇总",
          link:"http://news.mtc.sohu.com/news/channel/4/news.rss",
          desc:"erewr"
        },
        {
          title:"国际足坛",
          link:"http://news.mtc.sohu.com/news/channel/5/news.rss",
          desc:"werewr"
        }
        ]
      }

      ]
    }
}
```

这段JSON描述了一个对象（最外层大括号包围的部分），为了方便区分，我们就将其称为对象A吧。对象A有两个Item（即key-value键值对），一个是result，其值为true；一个是root，其值为一个对象，称为对象B。对象B也有两个Item，一个是version，其值为一个字串”201007091640”；一个是channels，其值是一个数组，而数组的成员都是一个象，每个对象又包含两个Item，一个是name，值分别为字串"新闻中心"和"体育新闻"；一个是subchnls，值都是数组个数组又分别有若干个成员，每个subchnls成员也都是一个对象，每个对象都有三个Item：title、link和desc。也许你看这，已经是一头大汗了，不过没关系，我们来帖张这段JSON文本对应的结构图，有图就有真相，请看图六：



（图六：黑色实线为对象，虚线为值，橙色实线为数组）

在RssReader中使用cJSON：

在RssReader中使用了开源库cJSON来解析JSON，所以在此就介绍下cJSON的使用：

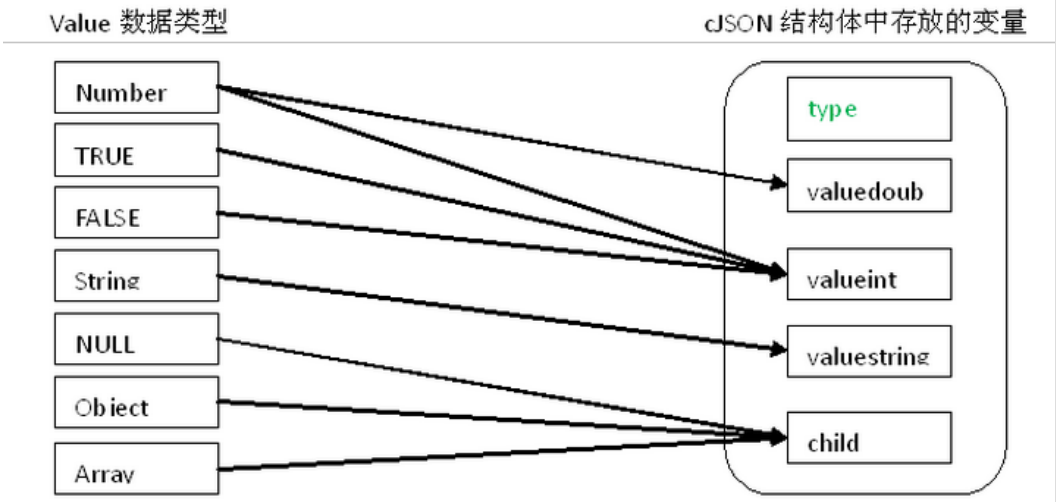
在CJSON中，一个key-value键值对被解析并存放在一个cJSON结构体变量中，其value取值集为：FALSE，TRUE，NUL
NUMBER，STRING，OBJECT，ARRAY。它们分别被存放在CJSON对象的child、valuestring、valueint、valuedouble变
中，而用于判断某个CJSON对象value的数据类型则是CJSON对象的type变量，其取值范围与CJSON对象的value集是一
对应的，如：cJSON_False对应FALSE。
cJSON Types:

```
#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
```

cJSON 结构体:

```
typedef struct cJSON
{
   struct cJSON *next,*prev;   //指向上一项/下一项
   struct cJSON *child;    //指向下一级，也就是当type为cJSON_Object或cJSON_Array时，此指针不为空。
   int type;
   char *valuestring;  // 当type为cJSON_String时
   int valueint;       // 当 type为cJSON_Number时
   double valuedouble; //当type为cJSON_Number时

   char *string;       // 当前项的名称，也就是key-value键值对的key
} cJSON;
```

在解析JSON过程中，从JSON格式描述的value数据到CJSON对象中存放的变量的一个映射关系如图七：



（图七）

对CJSON格式的解析是使用cJSON_Parse()方法，其传入的参数是一个CJSON的Object/Array结构的字串，解析成功则
一个cJSON结构体变量的指针，在使用完成后需要调用cJSON_Delete()将该指针销毁。CJSON是以树状结构来组织内部
各个cJSON结构体变量的，一般地，要使用某个cJSON结构体变量，需要调用cJSON_GetObjectItem()方法并根据其父
的cJSON结构体变量指针与该项的名称来获取其指针，举个例子：

```
bool bResult;
char jsonString[] = "{result:true}";
//获取result的值true
cJSON* pItem = NULL;
cJSON* pRoot = cJSON_Parse ( jsonString );
if ( pRoot )
{
   pItem = cJSON_GetObjectItem ( pRoot, "result" );
   if ( pItem )
   {
      bResult = pItem->valueint;   //由于result的值type为cJSON_False或cJSON_True，所以其值被存放在valueint变量
   }
   cJSON_Delete ( pRoot );
}
```

在上例中，不管result的值type为何类型，都是通过调用cJSON_GetObjectItem()方法获取其对应的cJSON结构体变量的
针，只是在处理其对应的值时会有所不同。如果result的值type为cJSON_Object，则需要通过调用
cJSON_GetObjectItem(pItem, "subItemKey")来获取其子Item的指针。在处理值type为cJSON_Array的Item时，就需要
用到另外两个API：cJSON_GetArraySize ()和cJSON_GetArrayItem()。我们举个获取一个数组成员值的例子：

```
char* out;
char jsonString[] = "{colors:[\"red\", \"green\",\ "blue\", \"yellow\", \"white\"]}";
cJSON* pArray = NULL;
cJSON* pRoot = cJSON_Parse ( jsonString );
if ( pRoot )
{
   pArray = cJSON_GetObjectItem ( pRoot, "colors" );
   if ( pArray )
   {
      cJSON* pArrayItem = NULL;
      int nCount = cJSON_GetArraySize ( pArray ); //获取pArray数组的大小
      for( int i = 0; i < nCount; i++)
      {
         pArrayItem = cJSON_GetArrayItem(pArray, i);
         out = cJSON_Print( pArrayItem );   //将pArrayItem的值以字串的形式打印到char型buffer上，cJSON_Print()会
分配内存空间，用完需要释放内存。
         SS_printf( "array item %d: %s\n", i, out);
         Free( out );
      }
   }
   cJSON_Delete ( pRoot );
```

```
}
```

在提取一个复杂的JSON格式的数据时，也只是将以上两个例子使用到的方法进行组合调用罢了。所以对CJSON提供的的使用是很简单有效的。有了以上知识的了解，我们就可以编写一些代码将例一中的JSON解析并提取其中的数据，还点代码才是硬道理，代码如下：

TChannelsData.h:

```
/** 子频道信息结构体
*
*/
struct SUBCHNL_DATA
{
    SUBCHNL_DATA();
    void clear();

    TUChar * m_title;
    char * m_link;
    TUChar * m_desc;
};

/** 大频道信息结构体
*
*/
struct CHANNEL_DATA
{
    CHANNEL_DATA();

    TUChar* m_pszTitle;
    vector m_aSubChnlList;
};

//.............
// TChannelsData 类成员变量：RSSReaderConfig 版本号
char m_pszVersion[32];
// TChannelsData 类成员变量：频道信息列表
vector m_aChnlsList;
//.............
```

TChannelsData.cpp:

```
/** 解析JSON格式的内容
*
* \param pszJsonText 解析的JSON格式内容字串
*
* \return true:有更新数据; false:没有更新数据
*/
Boolean TChannelsData::ParseJson(char* pszJsonText)
{
    //char* out;
    cJSON* objJson;

    objJson= cJSON_Parse(pszJsonText);

    if (objJson)
    {
        //out=cJSON_Print(objJson);
        cJSON* objRootItem = NULL;

        //判断是否需要更新
        objRootItem = cJSON_GetObjectItem(objJson, "result");
        if (objRootItem)
        {
            if (!objRootItem->valueint)
            {
                return FALSE;
            }
        }
        else
```

```
        {
            return FALSE;
        }

        //获取更新数据，根节点root
        objRootItem = cJSON_GetObjectItem(objJson, "root");
        if (objRootItem)
        {
            cJSON* objJsonItem = NULL;

            //获取版本号
            objJsonItem = cJSON_GetObjectItem(objRootItem, "version");
            if (objJsonItem)
            {
                Int32 nLen = strlen(objJsonItem->valuestring);
                strncpy(m_pszVersion, objJsonItem->valuestring, (nLen < 32)? nLen : 31);
            }

            //解析出大频道
            _ParseChannels(objRootItem);
        }

        //SS_printf("=======[parse json]%s\n",out);
        cJSON_Delete(objJson);
        //free(out);
    }

    return TRUE;
}

/** 解析出大频道
*
* \param pCJson cJSON对象指针
*
* \return void
*/
void TChannelsData::_ParseChannels(cJSON* pCJson)
{
    cJSON* pJsonArray = NULL;

    if (!pCJson)
    {
        return;
    }

    pJsonArray = cJSON_GetObjectItem(pCJson, "channels");
    if(pJsonArray)
    {
        cJSON* pArrayItem = NULL;
        cJSON* pJsonTemp = NULL;

        Int32 nSize = cJSON_GetArraySize(pJsonArray);
        for (Int32 i = 0; i < nSize; i++)
        {
            pArrayItem = cJSON_GetArrayItem(pJsonArray, i);
            if (pArrayItem)
            {
                CHANNEL_DATA tChannelData;
                Int32 nLen = 0;

                //获取大频道名称
                tChannelData.m_pszTitle = _JsonGetTUString(pArrayItem, "name");

                //解析出子频道
                _ParseSubChnls(tChannelData.m_aSubChnlList, pArrayItem);

                //将大频道信息对象压入列表中
```

```
            m_aChnlsList.push_back(tChannelData);
        }
        else
        {
            continue;
        }
    }
  }
}

/**  解析子频道
*
* \param aSubChnlList 存放子频道数据的vector对象
* \param pCJson cJSON对象指针
*
* \return void
*/
void TChannelsData::_ParseSubChnls(vector& aSubChnlList, cJSON* pCJson)
{
  cJSON* pJsonArray = NULL;

  if (!pCJson)
  {
    return;
  }

  pJsonArray = cJSON_GetObjectItem(pCJson, "subchnls");
  if (pJsonArray)
  {
    cJSON* pArrayItem = NULL;
    //cJSON* pJsonTemp = NULL;

    Int32 nSize = cJSON_GetArraySize(pJsonArray);
    for (Int32 i = 0; i < nSize; i++)
    {
      pArrayItem = cJSON_GetArrayItem(pJsonArray, i);
      if (pArrayItem)
      {
        SUBCHNL_DATA tSubChnlData;
        Int32 nLen = 0;

        //get title
        tSubChnlData.m_title = _JsonGetTUString(pArrayItem, "title");

        //get link
        tSubChnlData.m_link = _JsonGetString(pArrayItem, "link");

        //get desc
        tSubChnlData.m_desc = _JsonGetTUString(pArrayItem, "desc");

        aSubChnlList.push_back(tSubChnlData);
      }
    }
  }
}

/** 获取指定的cJSON对象的指定属性值
*
* \param pJsonItem cJSON对象指针
* \param pszKey cJSON对象属性
*
* \return 返回JSON对象的值，以TUChar字串形式返回
*/
TUChar* TChannelsData::_JsonGetTUString(cJSON* pJsonItem, char* pszKey)
{
  TUChar* pszValue = NULL;
  Int32 nLen;
```

```
    cJSON* pJsonTemp = NULL;

    pJsonTemp = cJSON_GetObjectItem(pJsonItem, pszKey);
    if (pJsonTemp)
    {
        nLen = strlen(pJsonTemp->valuestring) + 1;
        pszValue = new TUChar[nLen];
        if(pszValue)
        {
            MemSet(pszValue, 0, nLen * sizeof(TUChar));
            TUString::StrUtf8ToStrUnicode(pszValue, (const Char*)pJsonTemp->valuestring);
        }
    }

    return pszValue;
}

/** 获取指定的cJSON对象的指定属性值
*
* \param pJsonItem cJSON对象指针
* \param pszKey cJSON对象属性
*
* \return 返回JSON对象的值，以char字串形式返回
*/
char* TChannelsData::_JsonGetString(cJSON* pJsonItem, char* pszKey)
{
    char* pszValue = NULL;
    Int32 nLen;
    cJSON* pJsonTemp = NULL;

    pJsonTemp = cJSON_GetObjectItem(pJsonItem, pszKey);
    if (pJsonTemp)
    {
        nLen = strlen(pJsonTemp->valuestring) + 1;
        pszValue = new char[nLen];
        if(pszValue)
        {
            MemSet(pszValue, 0, nLen);
            strncpy(pszValue, pJsonTemp->valuestring, nLen - 1);
        }
    }

    return pszValue;
}

/** 获取指定的cJSON对象的指定属性值
*
* \param pJsonItem cJSON对象指针
* \param pszKey cJSON对象属性
*
* \return 返回JSON对象的值，以int32形式返回
*/
Int32 TChannelsData::_JsonGetInt(cJSON* pJsonItem, char* pszKey)
{
    Int32 nValue = 0;
    cJSON* pJsonTemp = NULL;

    pJsonTemp = cJSON_GetObjectItem(pJsonItem, pszKey);
    if (pJsonTemp)
    {
        nValue = pJsonTemp->valueint;
    }

    return nValue;
}

/** 获取指定的cJSON对象的指定属性值
```

```
 *
 * \param pJsonItem cJSON对象指针
 * \param pszKey cJSON对象属性
 *
 * \return 返回JSON对象的值，以Boolean形式返回
 */
Boolean TChannelsData::_JsonGetBoolean(cJSON* pJsonItem, char* pszKey)
{
    Boolean bValue = FALSE;
    cJSON* pJsonTemp = NULL;

    pJsonTemp = cJSON_GetObjectItem(pJsonItem, pszKey);
    if (pJsonTemp)
    {
        if(pJsonTemp->valueint)
        {
            bValue = TRUE;
        }
    }

    return bValue;
}
```

总结：

JSON的结构简约，所以使得JSON的文档的数据量比较小，比较适合用于网络数据的交换，而且对JSON文档的解析和提取的方法也很简单，方便程序员的使用，当然也正是因为JSON的结构简约，使得JSON的可读性与可编辑性会稍差于XML，所以JSON比较适合在较少有人工阅读和编辑的情况下使用期。

备注：经验证名称需加\" 比如char jsonString[] = "{\"result\":true}";

更多 0

上一篇：Linux下Eclipse编译时，报recompile with -fPIC错误，解决方法

下一篇：cJson 创建 读取

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

专区推荐内容

Javascript：this用...
Windows 8 | XAML...
利用IA SIMD技术来加速游戏...
英特尔软件开发工具
超极本触摸屏一种全新的动手方式
[Intel CPU中的安全指令... << >>

更多招聘职位 我公司职位也要出现在这里

【数赢云网络科技有限公司】Java程序员
【逻辑适点平面设计（北京）有限公司】软件开发工程师
【宜信互联网】技术经理（移动互联网）
【Wicresoft】网页重构工程师
【蓝谷】技术总监
【广州鸿根信息科技有限公司】VC++高级软件工程师

核心技术类目

全部主题 数据挖掘 SOA UML 开放平台 HTML5 开源 移动开发 iOS Android 移动游戏
Windows Phone JavaScript CSS 游戏引擎 云计算 大数据 Hadoop OpenStack 云平台 PHP
MongoDB JSON Xcode Node.js 前端开发 神经网络 安全 Java .NET MySQL textview
BigTable web框架 SQL Redis CouchDB Linux 可穿戴计算 NoSQL Ruby API GPL
XAML ASP.NET 前端开发 虚拟化 框架 机器学习 数据中心 IE10 敏捷 集群

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告