

冬冬他哥哥

博客园 首页 新随笔 联系 订阅 管理

随笔 - 448 文章 - 0 评论 - 23 阅读 - 56万

公告

昵称：谢呈勛
园龄：7年7个月
粉丝：78
关注：28
+加关注

搜索

找我看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

C#(6)
wpf(4)
NSIS(2)
image(2)
linux(2)
qt(2)
C#调用C++(2)
平台调用(2)
wxs(2)
win10(1)
更多

随笔分类

C#(67)
C#异步编程(22)
C#中的事件(6)
C#中的委托(4)
C++(101)
C++实现设计模式(1)

Http协议之libcurl实现

http协议之详解 (点我)

http协议之https (点我)

http协议之libcurl (点我)

一、libcurl简介

libcurl是一个跨平台的网络协议库，支持http, https, ftp, gopher, telnet, dict, file, 和ldap 协议。libcurl同样支持HTTPS证书授权，HTTP POST, HTTP PUT, FTP 上传，HTTP基本表单上传，代理，cookies,和用户认证。

libcurl的官网 <http://curl.haxx.se/>

库下载https://github.com/curl/curl/releases/tag/curl-7_71_1

二、libcurl的使用

调用curl_global_init()初始化libcurl

调用curl_easy_init()函数得到 easy interface型指针

调用curl_easy_setopt()设置传输选项

根据curl_easy_setopt()设置的传输选项，实现回调函数以完成用户特定任务

调用curl_easy_perform()函数完成传输任务

调用curl_easy_cleanup()释放内存

三、函数简介

1.CURLcode curl_global_init(long flags);函数只能用一次。(其实在调用curl_global_cleanup 函数后仍然可再用)

如果这个函数在curl_easy_init函数调用时还没调用，它讲由libcurl库自动调用，所以多线程下最好主动调用该函数以防止在线程中curl_easy_init时多次调用。

注意：虽然libcurl是线程安全的，但curl_global_init是不能保证线程安全的，所以不要在每个线程中都调用curl_global_init，应该将该函数的调用放在主线程中。
参数：flags

CURL_GLOBAL_ALL //初始化所有的可能的调用。
CURL_GLOBAL_SSL //初始化支持 安全套接字层。
CURL_GLOBAL_WIN32 //初始化win32套接字层。
CURL_GLOBAL_NOHING //没有额外的初始化。

2、void curl_global_cleanup(void);
结束libcurl使用的时候，用来对curl_global_init做的工作清理。类似于close的函数。

注意：虽然libcurl是线程安全的，但curl_global_cleanup是不能保证线程安全的，所以不要在每个线程中都调用curl_global_init，应该将该函数的调用放在主线程中。

3 char *curl_version();
打印当前libcurl库的版本。

https://www.cnblogs.com/xietianjiao/p/13260021.html

1/13

COM技术(4)
C语言(1)
Docker技术(9)
Linux(16)
Nodejs(3)
NSIS(6)
QT(3)
react框架开发(3)
RPA(1)
更多

随笔档案

2023年3月(5)
2023年2月(1)
2022年12月(1)
2022年11月(5)
2022年10月(4)
2022年9月(11)
2022年4月(3)
2022年3月(2)
2022年2月(2)
2022年1月(5)
2021年12月(4)
2021年11月(2)
2021年10月(6)
2021年9月(14)
2021年8月(6)
更多

阅读排行榜

1. c++引用lib和dll的方法总结(21977)
2. LPCTSTR 用法(13780)
3. C#中的异步和同步(13545)
4. C# Task的暂停与终止(12582)
5. c++11的异步方法 及线程间通信(12312)

评论排行榜

1. .Net环境下调用ProtoBuf(5)
2. c++的urlmon实现下载文件并进度回调(2)
3. .Net的内存回收(1)
4. WPF 中ContextMenu 在mvvm模式中的绑定问题(1)
5. WPF的动态资源和静态资源(1)

推荐排行榜

1. wpf的uri (让你完全明白wpf的图片加载方式以及URI写法) (4)
2. WPF自定义控件 (一) の控件分类 (3)

4 CURL *curl_easy_init();
curl_easy_init用来初始化一个CURL的指针(有些像返回FILE类型的指针一样). 相应的在调用结束时要用curl_easy_cleanup函数清理.
一般curl_easy_init意味着一个会话的开始. 它会返回一个easy_handle(CURL*对象), 一般都用在easy系列的函数中.

5 void curl_easy_cleanup(CURL *handle);
这个调用用来结束一个会话.与curl_easy_init配合着用.
参数:
CURL类型的指针.

6 CURLcode curl_easy_setopt(CURL *handle, CURLOPToption option, parameter);
这个函数最重要了.几乎所有的curl 程序都要频繁的使用它.它告诉curl库.程序将有如何的行为. 比如要查看一个网页的html代码等.(这个函数有些像ioctl函数)参数:
1 CURL类型的指针
2 各种CURLOption类型的选项.(都在curl.h库里有定义,man 也可以查看到)
3 parameter 这个参数 既可以是个函数的指针,也可以是某个对象的指针,也可以是个long型的变量.它用什么这取决于第二个参数.
CURLOption 这个参数的取值很多.具体的可以查看man手册.

7 CURLcode curl_easy_perform(CURL *handle);

在初始化CURL类型的指针 以及curl_easy_setopt完成后调用. 就像字面的意思所说perform就像是 个舞台.让我们设置的 option 运作起来.参数:
CURL类型的指针.

三、 curl_easy_setopt函数部分选项介绍
本节主要介绍curl_easy_setopt中跟http相关的参数。该函数是curl中非常重要的函数， curl所有设置都是在该函数中完成的，该函数的设置选项众多，注意本节的阐述的只是部分常见选项。

1. CURLOPT_URL
设置访问URL

2. CURLOPT_WRITEFUNCTION, CURLOPT_WRITEDATA
回调函数原型为: size_t function(void *ptr, size_t size, size_t nmemb, void *stream); 函数将在libcurl接收到数据后被调用，因此函数多做数据保存的功能，如处理下载文件。
CURLOPT_WRITEDATA 用于表明CURLOPT_WRITEFUNCTION函数中的stream指针的来源。

如果你没有通过CURLOPT_WRITEFUNCTION属性给easy handle设置回调函数，libcurl会提供一个默认的回调函数，它只是简单的将接收到的数据打印到标准输出。你也可以通过CURLOPT_WRITEDATA属性给默认回调函数传递一个已经打开的文件指针，用于将数据输出到文件里。

3. CURLOPT_HEADERFUNCTION, CURLOPT_HEADERDATA
回调函数原型为 size_t function(void *ptr, size_t size,size_t nmemb, void *stream); libcurl一旦接收到http 头部数据后将调用该函数。CURLOPT_WRITEDATA 传递指针给libcurl，该指针表明CURLOPT_HEADERFUNCTION 函数的stream指针的来源。

4. CURLOPT_READFUNCTION CURLOPT_READDATA
libCurl需要读取数据传递给远程主机时将调用CURLOPT_READFUNCTION指定的函数，函数原型是: size_t function(void *ptr, size_t size, size_t nmemb,void *stream). CURLOPT_READDATA 表明CURLOPT_READFUNCTION函数原型中的stream指针来源。

- 3. C# AOP的实现 (利用.Net自带的轻量级框架RealProxy) (2)
- 4. WPF可视对象变换 (RenderTransform) -----RotateTransform、TranslateTransform、ScaleTransform(2)
- 5. c++智能指针 (unique_ptr、shared_ptr、weak_ptr、auto_ptr) (2)

最新评论

- 1. Re:C# Task的暂停与终止
if
(_cancellationToken.IsCancellationRequested) { return; } 如果后面的逻辑需要执行10分钟猜完毕，岂不是要等待10分钟猜能正在取消？这个cancelIT...
--wgscd
- 2. Re:WPF的动态资源和静态资源
666 这份介绍的太详细了
--彭二狗的牵引绳
- 3. Re:WPF 中ContextMenu 在mvvm模式中的绑定问题
您好！请问你下这个上下文动态绑定的有没有源码，可以发一份吗？
564178640@qq.com
--天空之上的鱼儿
- 4. Re:WPF自定义控件 (四) の自定义控件
能把源码发一下吗？多谢
--zhhu
- 5. Re:.Net的内存回收
string是引用类型，所有引用类型的默认值都是null。
--羽烈

- 5. CURLOPT_NOPROGRESS, CURLOPT_PROGRESSFUNCTION, CURLOPT_PROGRESSDATA
跟数据传输进度相关的参数。CURLOPT_PROGRESSFUNCTION 指定的函数正常情况下每秒被libcurl调用一次，为了使CURLOPT_PROGRESSFUNCTION被调用，CURLOPT_NOPROGRESS必须被设置为false，CURLOPT_PROGRESSDATA指定的参数将作为CURLOPT_PROGRESSFUNCTION指定函数的第一个参数
- 6. CURLOPT_TIMEOUT, CURLOPT_CONNECTIONTIMEOUT:
CURLOPT_TIMEOUT 由于设置传输时间，CURLOPT_CONNECTIONTIMEOUT 设置连接等待时间
- 7. CURLOPT_FOLLOWLOCATION
设置重定位URL
- 8. CURLOPT_RANGE: CURLOPT_RESUME_FROM:
断点续传相关设置。CURLOPT_RANGE 指定char *参数传递给libcurl，用于指明http域的RANGE头域，例如：
表示头500个字节：bytes=0-499
表示第二个500字节：bytes=500-999
表示最后500个字节：bytes=-500
表示500字节以后的范围：bytes=500-
第一个和最后一个字节：bytes=0-0,-1
同时指定几个范围：bytes=500-600,601-999
CURLOPT_RESUME_FROM 传递一个long参数给libcurl，指定你希望开始传递的 偏移量。

四、curl_easy_perform 函数说明 (error 状态码)
该函数是完成curl_easy_setopt指定的所有选项，本节重点介绍curl_easy_perform的返回值。返回0意味一切ok，非0代表错误发生。主要错误码说明：

- 1. CURLE_OK
任务完成一切都好
- 2. CURLE_UNSUPPORTED_PROTOCOL
不支持的协议，由URL的头部指定
- 3. CURLE_COULDNT_CONNECT
不能连接到remote 主机或者代理
- 4. CURLE_REMOTE_ACCESS_DENIED
访问被拒绝
- 5. CURLE_HTTP_RETURNED_ERROR
Http返回错误
- 6. CURLE_READ_ERROR
读本地文件错误

要获取详细的错误描述字符串，可以通过const char *curl_easy_strerror(CURLcode errornum) 这个函数取得。

五、设置Http请求头

当使用libcurl发送http请求时，它会自动添加一些http头。我们可以通过CURLOPT_HTTPHEADER属性手动替换、添加或删除相应 的HTTP消息头。



Host
http1.1 (大部分http1.0) 版本都要求客户端请求提供这个信息头。

```
Pragma
"no-cache"。表示不要缓冲数据。
Accept
"*/*"。表示允许接收任何类型的数据。
Expect
```



以POST的方式向HTTP服务器提交请求时，libcurl会设置该消息头为"100-continue"，它要求服务器在正式处理该请求之前，返回一个"OK"消息。如果POST的数据很小，libcurl可能不会设置该消息头。

自定义选项

当前越来越多的协议都构建在HTTP协议之上（如：soap），这主要归功于HTTP的可靠性，以及被广泛使用的代理支持（可以穿透大部分防火墙）。这些协议的使用方式与传统HTTP可能有很大的不同。对此，libcurl作了很好的支持。

自定义请求方式(CustomRequest)

HTTP支持GET, HEAD或者POST提交请求。可以设置CURLOPT_CUSTOMREQUEST来设置自定义的请求方式，libcurl默认以GET方式提交请求：

```
curl_easy_setopt(easy_handle, CURLOPT_CUSTOMREQUEST, "MYOWNREQUEST");
```

修改消息头

HTTP协议提供了消息头，请求消息头用于告诉服务器如何处理请求；响应消息头则告诉浏览器如何处理接收到的数据。



```
struct curl_slist *headers=NULL; /* init to NULL is important */
headers = curl_slist_append(headers, "Hey-server-hey: how are you?");
headers = curl_slist_append(headers, "X-silly-content: yes");
/* pass our list of custom made headers */
curl_easy_setopt(easyhandle, CURLOPT_HTTPHEADER, headers);
curl_easy_perform(easyhandle); /* transfer http */
curl_slist_free_all(headers); /* free the header list */
```



对于已经存在的消息头，可以重新设置它的值：

```
headers = curl_slist_append(headers, "Accept: Agent-007");
headers = curl_slist_append(headers, "Host: munged.host.line");
```

删除消息头

对于一个已经存在的消息头，设置它的内容为空，libcurl在发送请求时就不会同时提交该消息头：

```
headers = curl_slist_append(headers, "Accept:");
```

六、获取http应答头信息

发出http请求后，服务器会返回应答头信息和应答数据，如果仅仅是打印应答头的所有内容，则直接可以通过curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, 打印函数)的方式来完成，这里需要获取的是应答头中特定的信息，比如应答码、cookies列表等，则需要通过下面这个函数：

```
CURLcode curl_easy_getinfo(CURL *curl, CURLINFO info, ... );
```

info参数就是我们需要获取的内容，下面是一些参数值：

```
1.CURLINFO_RESPONSE_CODE
获取应答码

2.CURLINFO_HEADER_SIZE
头大小

3.CURLINFO_COOKIELIST
```

cookies列表

除了获取应答信息外，这个函数还能获取curl的一些内部信息，如请求时间、连接时间等等。

七、多线程问题

首先一个基本原则就是：绝对不应该在线程之间共享同一个libcurl handle(CURL *对象)，不管是easy handle还是multi handle（本文只介绍easy_handle）。一个线程每次只能使用一个handle。

libcurl是线程安全的，但有两点例外：信号(signals)和SSL/TLS handler。信号用于超时失效名字解析(timing out name resolves)。libcurl依赖其他的库来支持SSL/STL，所以用多线程的方式访问HTTPS或FTPS的URL时，应该满足这些库对多线程操作的一些要求。详细可以参考：

OpenSSL: <http://www.openssl.org/docs/crypto/threads.html#DESCRIPTION>

GnuTLS: http://www.gnu.org/software/gnutls/manual/html_node/Multi_002dthreaded-applications.html

八、什么时候libcurl无法正常工作

传输失败总是有原因的。你可能错误的设置了一些libcurl的属性或者没有正确的理解某些属性的含义，或者是远程主机返回一些无法被正确解析的内容。

这里有一个黄金法则来处理这些问题：将CURLOPT_VERBOSE属性设置为1，libcurl会输出通信过程中的一些细节。如果使用的是http协议，请求头/响应头也会被输出。将CURLOPT_HEADER设为1，这些头信息将出现在消息的内容中。

当然不可否认的是，libcurl还存在bug。

如果你对相关的协议了解越多，在使用libcurl时，就越不容易犯错。

九、关于密码

客户端向服务器发送请求时，许多协议都要求提供用户名与密码。libcurl提供了多种方式来设置它们。

一些协议支持在URL中直接指定用户名和密码，类似于：

protocol://user:password@example.com/path/。libcurl能正确的识别这种URL中的用户名与密码并执行相应的操作。如果你提供的用户名和密码中有特殊字符，首先应该对其进行URL编码。

也可以通过CURLOPT_USERPWD属性来设置用户名与密码。参数是格式如“user:password”的字符串：

```
curl_easy_setopt(easy_handle, CURLOPT_USERPWD, "user_name:password");
```

有时候在访问代理服务器的时候，可能时时要求提供用户名和密码进行用户身份验证。这种情况下，libcurl提供了另一个属性CURLOPT_PROXYUSERPWD：

```
curl_easy_setopt(easy_handle, CURLOPT_PROXYUSERPWD, "user_name:password");
```

在UNIX平台下，访问FTP的用户名和密码可能会被保存在\$HOME/.netrc文件中。libcurl支持直接从这个文件中获取用户名与密码：

```
curl_easy_setopt(easy_handle, CURLOPT_NETRC, 1L);
```

在使用SSL时，可能需要提供一个私钥用于数据安全传输，通过CURLOPT_KEYPASSWD来设置私钥：

```
curl_easy_setopt(easy_handle, CURLOPT_KEYPASSWD, "keypassword");
```

十、HTTP验证

在使用HTTP协议时，客户端有很多种方式向服务器提供验证信息。默认的 HTTP验证方法是"Basic"，它将用户名与密码以明文的方式、经Base64编码后保存在HTTP请求头中，发往服务器。当然这不太安全。

当前版本的libcurl支持的验证方法有：basic, Digest, NTLM, Negotiate, GSS-Negotiate and SPNEGO。（译者感叹：搞Web这么多年，尽然不知道这些Http的验证方式，实在惭愧。）可以通过CURLOPT_HTTPAUTH属性来设置具体的验证方式：

```
curl_easy_setopt(easy_handle, CURLOPT_HTTPAUTH, CURLAUTH_DIGEST);
```

向代理服务器发送验证信息时，可以通过CURLOPT_PROXYAUTH设置验证方式：

```
curl_easy_setopt(easy_handle, CURLOPT_PROXYAUTH, CURLAUTH_NTLM);
```

也可以同时设置多种验证方式（通过按位与），使用 'CURLAUTH_ANY' 将允许libcurl可以选择任何它所支持的验证方式。通过CURLOPT_HTTPAUTH或 CURLOPT_PROXYAUTH属性设置的多种验证方式，libcurl会在运行时选择一种它认为是最好的方式与服务器通信：

```
curl_easy_setopt(easy_handle, CURLOPT_HTTPAUTH,
CURLAUTH_DIGEST|CURLAUTH_BASIC);
// curl_easy_setopt(easy_handle, CURLOPT_HTTPAUTH, CURLAUTH_ANY);
```

十一、代码示例

1.基本的http GET/POST操作

```

#include <stdio.h>
#include <curl/curl.h>
bool getUrl(char *filename)
{
    CURL *curl;
    CURLcode res;
    FILE *fp;
    if ((fp = fopen(filename, "w")) == NULL) // 返回结果用文件存储
        return false;
    struct curl_slist *headers = NULL;
    headers = curl_slist_append(headers, "Accept: Agent-007");
    curl = curl_easy_init(); // 初始化
    if (curl)
    {
        //curl_easy_setopt(curl, CURLOPT_PROXY, "10.99.60.201:8080");// 代理
        curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);// 改协议头
        curl_easy_setopt(curl, CURLOPT_URL, "http://www.baidu.com");
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, fp); //将返回的http头输出到fp指向
        curl_easy_setopt(curl, CURLOPT_HEADERDATA, fp); //将返回的html主体数据输出到
        res = curl_easy_perform(curl); // 执行
        if (res != 0) {

            curl_slist_free_all(headers);
            curl_easy_cleanup(curl);

        }
        fclose(fp);
        return true;
    }
}

bool postUrl(char *filename)
{
    CURL *curl;
    CURLcode res;
    FILE *fp;
    if ((fp = fopen(filename, "w")) == NULL)
```

```

        return false;
    }
    curl = curl_easy_init();
    if (curl)
    {
        curl_easy_setopt(curl, CURLOPT_COOKIEFILE, "/tmp/cookie.txt"); // 指定cookie文件
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, "&logintype=uid&u=xieyan&psw=123456"); // 指定post数据
        //curl_easy_setopt(curl, CURLOPT_PROXY, "10.99.60.201:8080");
        curl_easy_setopt(curl, CURLOPT_URL, "http://mail.sina.com.cn/cgi-bin/login.cgi");
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_callback);
        res = curl_easy_perform(curl);
        curl_easy_cleanup(curl);
    }
    fclose(fp);
    return true;
}

int main(void)
{
    getUrl("/tmp/get.html");
    postUrl("/tmp/post.html");
}

```



2 获取html网页

```

#include <stdio.h>
#include <curl/curl.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    CURL *curl; // 定义CURL类型的指针
    CURLcode res; // 定义CURLcode类型的变量，保存返回状态码

    if(argc!=2)
    {
        printf("Usage : file <url>;\n");
        exit(1);
    }


    curl = curl_easy_init(); // 初始化一个CURL类型的指针
    if(curl!=NULL)
    {
        // 设置curl选项。其中CURLOPT_URL是让用户指定url。argv[1]中存放的命令行传进来的url
        curl_easy_setopt(curl, CURLOPT_URL, argv[1]);
        // 调用curl_easy_perform 执行我们的设置，并进行相关的操作。在这里只在屏幕上显示出结果
        res = curl_easy_perform(curl);
        // 清除curl操作。
        curl_easy_cleanup(curl);
    }

    return 0;
}

```



3 网页下载保存实例



```
// 采用CURLOPT_WRITEFUNCTION 实现网页下载保存功能
#include <stdio.h>;
#include <stdlib.h>;
#include <unistd.h>;

#include <curl/curl.h>;
#include <curl/types.h>;
#include <curl/easy.h>;

FILE *fp; //定义FILE类型指针
//这个函数是为了符合CURLOPT_WRITEFUNCTION而构造的
//完成数据保存功能
size_t write_data(void *ptr, size_t size, size_t nmemb, void *stream)
{
    int written = fwrite(ptr, size, nmemb, (FILE *)fp);
    return written;
}


int main(int argc, char *argv[])
{
    CURL *curl;

    curl_global_init(CURL_GLOBAL_ALL);
    curl=curl_easy_init();
    curl_easy_setopt(curl, CURLOPT_URL, argv[1]);

    if((fp=fopen(argv[2], "w"))==NULL)
    {
        curl_easy_cleanup(curl);
        exit(1);
    }
    ///CURLOPT_WRITEFUNCTION 将后继的动作交给write_data函数处理
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_data);
    curl_easy_perform(curl);
    curl_easy_cleanup(curl);
    exit(0);
}
```



4 进度条实例显示文件下载进度



```
// 采用CURLOPT_NOPROGRESS, CURLOPT_PROGRESSFUNCTION, CURLOPT_PROGRESSDATA 实现
//函数采用了gtk库, 故编译时需指定gtk库
//函数启动专门的线程用于显示gtk 进度条bar
#include <stdio.h>
#include <gtk/gtk.h>
#include <curl/curl.h>
#include <curl/types.h> /* new for v7 */
#include <curl/easy.h> /* new for v7 */

GtkWidget *Bar;
```



```
////这个函数是为了符合CURLOPT_WRITEFUNCTION而构造的
//完成数据保存功能
size_t my_write_func(void *ptr, size_t size, size_t nmemb, FILE *stream)
{
    return fwrite(ptr, size, nmemb, stream);
}

//这个函数是为了符合CURLOPT_READFUNCTION而构造的
//数据上传时使用
size_t my_read_func(void *ptr, size_t size, size_t nmemb, FILE *stream)
{
    return fread(ptr, size, nmemb, stream);
}

//这个函数是为了符合CURLOPT_PROGRESSFUNCTION而构造的
//显示文件传输进度, t代表文件大小, d代表传 输已经完成部分
int my_progress_func(GtkWidget *bar,
                    double t, /* dltotal */
                    double d, /* dlnow */
                    double ultotal,
                    double ulnow)
{
    /* printf("%d / %d (%g %%)\n", d, t, d*100.0/t);*/
    gdk_threads_enter();
    gtk_progress_set_value(GTK_PROGRESS(bar), d*100.0/t);
    gdk_threads_leave();
    return 0;
}

void *my_thread(void *ptr)
{
    CURL *curl;
    CURLcode res;
    FILE *outfile;
    gchar *url = ptr;

    curl = curl_easy_init();
    if(curl)
    {
        outfile = fopen("test.curl", "w");

        curl_easy_setopt(curl, CURLOPT_URL, url);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, outfile);
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, my_write_func);
        curl_easy_setopt(curl, CURLOPT_READFUNCTION, my_read_func);
        curl_easy_setopt(curl, CURLOPT_NOPROGRESS, 0L);
        curl_easy_setopt(curl, CURLOPT_PROGRESSFUNCTION, my_progress_func);
        curl_easy_setopt(curl, CURLOPT_PROGRESSDATA, Bar);

        res = curl_easy_perform(curl);

        fclose(outfile);
        /* always cleanup */
        curl_easy_cleanup(curl);
    }

    return NULL;
}

int main(int argc, char **argv)
```

```

{
    GtkWidget *Window, *Frame, *Frame2;
    GtkAdjustment *adj;

    /* Must initialize libcurl before any threads are started */
    curl_global_init(CURL_GLOBAL_ALL);

    /* Init thread */
    g_thread_init(NULL);

    gtk_init(&argc, &argv);
    Window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    Frame = gtk_frame_new(NULL);
    gtk_frame_set_shadow_type(GTK_FRAME(Frame), GTK_SHADOW_OUT);
    gtk_container_add(GTK_CONTAINER(Window), Frame);
    Frame2 = gtk_frame_new(NULL);
    gtk_frame_set_shadow_type(GTK_FRAME(Frame2), GTK_SHADOW_IN);
    gtk_container_add(GTK_CONTAINER(Frame), Frame2);
    gtk_container_set_border_width(GTK_CONTAINER(Frame2), 5);
    adj = (GtkAdjustment*)gtk_adjustment_new(0, 0, 100, 0, 0, 0);
    Bar = gtk_progress_bar_new_with_adjustment(adj);
    gtk_container_add(GTK_CONTAINER(Frame2), Bar);
    gtk_widget_show_all(Window);

    if (!g_thread_create(&my_thread, argv[1], FALSE, NULL) != 0)
        g_warning("can't create the thread");

    gdk_threads_enter();
    gtk_main();
    gdk_threads_leave();
    return 0;
}

```



5 断点续传实例



```

//采用CURLOPT_RESUME_FROM_LARGE 实现文件断点续传功能
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>

#include <curl/curl.h>
//这个函数为CURLOPT_HEADERFUNCTION参数构造
/* 从http头部获取文件大小*/
size_t getcontentlengthfunc(void *ptr, size_t size, size_t nmemb, void *stream)
{
    int r;
    long len = 0;

    /* _snscanf() is Win32 specific */
    // r = _snscanf(ptr, size * nmemb, "Content-Length: %ld\n", &len);
    r = sscanf(ptr, "Content-Length: %ld\n", &len);
    if (r) /* Microsoft: we don't read the specs */
        *((long *) stream) = len;
}

```

```
        return size * nmemb;
    }

    /* 保存下载文件 */
    size_t wirtefunc(void *ptr, size_t size, size_t nmemb, void *stream)
    {
        return fwrite(ptr, size, nmemb, stream);
    }

    /*读取上传文件 */
    size_t readfunc(void *ptr, size_t size, size_t nmemb, void *stream)
    {
        FILE *f = stream;
        size_t n;

        if (ferror(f))
            return CURL_READFUNC_ABORT;

        n = fread(ptr, size, nmemb, f) * size;

        return n;
    }

    // 下载 或者上传文件函数
    int download(CURL *curlhandle, const char * remotepath, const char * localpath,
                long timeout, long tries)
    {
        FILE *f;
        curl_off_t local_file_len = -1 ;
        long filesize =0 ;

        CURLcode r = CURLE_GOT_NOTHING;
        int c;
        struct stat file_info;
        int use_resume = 0;
        /* 得到本地文件大小 */
        //if(access(localpath,F_OK) ==0)

        if(stat(localpath, &file_info) == 0)
        {
            local_file_len = file_info.st_size;
            use_resume = 1;
        }
        //采用追加方式打开文件，便于实现文件断点续传工作
        f = fopen(localpath, "ab+");
        if (f == NULL) {
            perror(NULL);
            return 0;
        }

        //curl_easy_setopt(curlhandle, CURLOPT_UPLOAD, 1L);

        curl_easy_setopt(curlhandle, CURLOPT_URL, remotepath);

        curl_easy_setopt(curlhandle, CURLOPT_CONNECTTIMEOUT, timeout); //
        //设置http 头部处理函数
        curl_easy_setopt(curlhandle, CURLOPT_HEADERFUNCTION, getcontentlengthfun
        curl_easy_setopt(curlhandle, CURLOPT_HEADERDATA, &filesize);
```

```
// 设置文件续传的位置给libcurl
curl_easy_setopt(curlhandle, CURLOPT_RESUME_FROM_LARGE, use_resume?local

curl_easy_setopt(curlhandle, CURLOPT_WRITEDATA, f);
curl_easy_setopt(curlhandle, CURLOPT_WRITEFUNCTION, wirtefunc);

//curl_easy_setopt(curlhandle, CURLOPT_READFUNCTION, readfunc);
//curl_easy_setopt(curlhandle, CURLOPT_READDATA, f);
curl_easy_setopt(curlhandle, CURLOPT_NOPROGRESS, 1L);
curl_easy_setopt(curlhandle, CURLOPT_VERBOSE, 1L);

r = curl_easy_perform(curlhandle);

fclose(f);

if (r == CURLE_OK)
    return 1;
else {
    fprintf(stderr, "%s\n", curl_easy_strerror(r));
    return 0;
}
}

int main(int c, char **argv) {
    CURL *curlhandle = NULL;

    curl_global_init(CURL_GLOBAL_ALL);
    curlhandle = curl_easy_init();

    //download(curlhandle, "ftp://user:pass@host/path/file", "C:\\file", 0,
    download(curlhandle , "http://software.sky-union.cn/index.asp", "/work/index.a
    curl_easy_cleanup(curlhandle);
    curl_global_cleanup();

    return 0;
}
```



[http协议之详解（点我）](#)

[http协议之https（点我）](#)

[http协议之libcurl（点我）](#)

分类: [网络](#)

好文要顶

关注我

收藏该文

谢呈勛

粉丝 - 78 关注 - 28

[+加关注](#)

0

0

« 上一篇: [Windows进程通信 \(IPC\) 之邮件槽](#)
» 下一篇: [C#线程同步方式](#)

posted @ 2020-07-07 12:08 谢呈勛 阅读(6322) 评论(0) 编辑 收藏 举报

[刷新评论](#) [刷新页面](#) [返回顶部](#)

发表评论

编辑预览

B🔗🔼🔽🔍

支持 Markdown

🔍 自动补全

[提交评论](#) [退出](#) [订阅评论](#)

[Ctrl+Enter快捷键提交]

【推荐】博客园人才出海服务第一站，联合日本好融社推出日本IT人才移民直通车
【推荐】中国云计算领导者：阿里云轻量应用服务器2核2G，新用户低至108元/年

编辑推荐:

- 从内核源码看 slab 内存池的创建初始化流程
- [C#表达式树] 最完善的表达式树 Expression.Dynamic 的玩法
- 有意思的气泡 Loading 效果
- Three.js 进阶之旅：全景漫游-高阶版在线看房
- 解 Bug 之路 - 应用 999 线升高

阅读排行:

- 园子的商业化努力-困境求助：开设捐助通道
- 程序员的哲学
- 【从零开始】Docker Desktop：听说你小子要玩我
- AI时代下普通小程序员的想法
- [MAUI]模仿微信“按住-说话”的交互实现