<https://blog.csdn.net/lijinqi1987/article/details/53925835>

libcurl主要提供了两种发送http请求的方式，分别是Easy interface方式和multi interface方式，前者是采用阻塞的方式发送单条数据，后者采用组合的方式可以一次性发送多条数据

一、Easy interface

libcurl的easy interface是最基本的用法，简要流程为：  
1、在主线程中调用curl\_global\_init(CURL\_GLOBAL\_ALL)初始化  
2、调用curl\_easy\_init获取一个句柄；  
3、调用curl\_easy\_setopt函数设置此次传输的一些基本参数，如url地址、http头、cookie信息、发送超时时间等，其中，CURLOPT\_URL是必设的选项；  
4、设置完成后，调用curl\_easy\_perform函数发送数据；  
5、数据发送完毕后，调用curl\_easy\_cleanup清空句柄；  
6、调用curl\_global\_cleanup()做清理工作。

实现代码：

**[cpp]** [view plain](https://blog.csdn.net/lijinqi1987/article/details/53925835) [copy](https://blog.csdn.net/lijinqi1987/article/details/53925835)

1. **bool** send\_easy\_hanler(**char**\* post\_url, req\_t\* req)
2. {
3. //easy handler的句柄
4. CURL\* curl = NULL;
5. CURLcode res = CURLE\_OK;
6. //HTTP报文头
7. **struct** curl\_slist\* headers = NULL;
8. **char** tmp\_str[256] = { 0 };
10. //构建HTTP报文头
11. snprintf(tmp\_str, **sizeof**(tmp\_str), "User-Agent: %s", req->user\_agent\_);
12. headers = curl\_slist\_append(headers, tmp\_str);
13. snprintf(tmp\_str, **sizeof**(tmp\_str), "Accept-Language: %s", req->language\_);
14. headers = curl\_slist\_append(headers, tmp\_str);
15. snprintf(tmp\_str, **sizeof**(tmp\_str), "X-FORWORDED-FOR: %s", req->ip\_.c\_str());
16. headers = curl\_slist\_append(headers, tmp\_str);
18. /\*这个函数只能用一次,如果这个函数在curl\_easy\_init函数调用时还没调用，
19. 它讲由libcurl库自动调用，所以多线程下最好在主线程中调用一次该函数以防止在线程
20. 中curl\_easy\_init时多次调用\*/
21. curl\_global\_init(CURL\_GLOBAL\_ALL);
23. //初始化easy handler句柄
24. curl = curl\_easy\_init();
25. **if** (curl) {
26. //设置post请求的url地址
27. curl\_easy\_setopt(curl, CURLOPT\_URL, post\_url);
28. //设置HTTP头
29. curl\_easy\_setopt(curl, CURLOPT\_HTTPHEADER, headers);
30. //设置发送超时时间
31. curl\_easy\_setopt(curl, CURLOPT\_TIMEOUT, 1);
33. //执行单条请求
34. res = curl\_easy\_perform(curl);
35. **if** (res != CURLE\_OK) {
36. //curl\_easy\_strerror进行出错打印
37. LOG(WARNING) << "curl\_easy\_perform() failed:" << curl\_easy\_strerror(res);
38. }
40. curl\_slist\_free\_all(headers);
42. //这个调用用来结束一个会话.与curl\_easy\_init配合着用
43. curl\_easy\_cleanup(curl);
45. //在结束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
46. curl\_global\_cleanup();
47. }

二、multi interface  
multi interface提供了多种easy interface没有的特性，主要是  
1、提供了pull接口，使用libcurl的程序能够决定何时何处调用libcurl来get/send数据  
2、在同一线程中实现多条数据同时发送，且并没有使得程序更加复杂  
3、程序可以在自己的文件描述符和curl的文件描述符中同时等待执行  
4、提供基于事件的处理、扩大传输规模到数千个并发连接  
  
multi接口的使用会比easy 接口稍微复杂点，毕竟multi接口是依赖easy接口的，简要流程为：  
1、使用curl\_multi\_init创建一个multi handle，这个handler会在后续的curl\_multi\_\*函数中使用multi handler可以同时并发传输多条数据，每一条单独的数据是由一个easy handler创建；  
2、需要事先将需要传输的所有easyhandler创建好，并使用curl\_easy\_setopt设置各自属性，接着调用curl\_multi\_add\_handle函数逐个添加到multi handle中；  
3、调用curl\_multi\_perform进程数据传输，传输过程中将会调用每一个easy handler设置的回调函数或者配置内容，程序通过函数curl\_multi\_fdset、select()提取信息来判断何时进行数据传输等操作，函数curl\_multi\_perform的一个输入参数储存仍在进行传输的数据量，通过读取该变量，可以判断multi handles是否运行完毕，传输完毕不代表传输成功，可能有一个或多个传输失败；  
4、调用函数curl\_multi\_info\_read可以获取当前或之前传输的信息，重复调用该函数直到该消息队列为空，每一条返回信息都包含对应的easl handler的传输情况；  
5、当一个easy handler传输完成，此easy handler仍然仍然停留在multi stack中，需要调用curl\_multi\_remove\_handle将其从multi stack中移除，然后调用curl\_easy\_cleanup将其关闭；  
6、当multi stack中的所有传输都完成时，调用 curl\_multi\_cleanup关闭multi handler，需要注意的是事先要调用curl\_easy\_cleanup逐个清空所有easy handler。

源码：

**[cpp]** [view plain](https://blog.csdn.net/lijinqi1987/article/details/53925835) [copy](https://blog.csdn.net/lijinqi1987/article/details/53925835)

1. #include <errno.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <curl/multi.h>
6. **static** **const** **char** \*urls[] = {
7. "http://www.microsoft.com",
8. "http://www.opensource.org",
9. "http://www.google.com",
10. "http://www.yahoo.com",
11. "http://www.ibm.com",
12. "http://www.mysql.com",
13. "http://www.oracle.com",
14. "http://www.ripe.net",
15. };
17. #define MAX 8 /\* number of simultaneous transfers \*/
18. #define CNT sizeof(urls)/sizeof(char\*) /\* total number of transfers to do \*/
20. /\*此函数读取libcurl发送数据后的返回信息，如果不设置此函数，
21. 那么返回值将会输出到控制台，影响程序性能\*/
22. **static** **size\_t** cb(**char** \*d, **size\_t** n, **size\_t** l, **void** \*p)
23. {
24. /\* take care of the data here, ignored in this example \*/
25. (**void**)d;
26. (**void**)p;
27. **return** n\*l;
28. }
30. //设置单个easy handler的属性添加单个easy handler到multi handler中，
31. **static** **void** init(CURLM \*cm, **int** i)
32. {
33. CURL \*eh = curl\_easy\_init();
35. curl\_easy\_setopt(eh, CURLOPT\_WRITEFUNCTION, cb);
36. curl\_easy\_setopt(eh, CURLOPT\_HEADER, 0L);
37. curl\_easy\_setopt(eh, CURLOPT\_URL, urls[i]);
38. curl\_easy\_setopt(eh, CURLOPT\_PRIVATE, urls[i]);
39. curl\_easy\_setopt(eh, CURLOPT\_VERBOSE, 0L);
41. //添加easy handler 到multi handler中
42. curl\_multi\_add\_handle(cm, eh);
43. }
45. **int** main(**void**)
46. {
47. CURLM \*cm;
48. CURLMsg \*msg;
49. **long** curl\_timeo;
50. unsigned **int** C=0;
51. **int** max\_fd, msgs\_left, still\_running = -1;//still\_running判断multi handler是否传输完毕
52. fd\_set fd\_read, fd\_write, fd\_except;
53. **struct** timeval T;
55. curl\_global\_init(CURL\_GLOBAL\_ALL);
57. cm = curl\_multi\_init();
59. //现在multi handler的最大连接数
60. curl\_multi\_setopt(cm, CURLMOPT\_MAXCONNECTS, (**long**)MAX);
62. **for**(C = 0; C < MAX; ++C) {
63. init(cm, C);
64. }

67. **do**{
68. curl\_multi\_perform(cm, &still\_running);
70. **if**(still\_running) {
71. FD\_ZERO(&fd\_read);
72. FD\_ZERO(&fd\_write);
73. FD\_ZERO(&fd\_except);
75. //获取multi curl需要监听的文件描述符集合 fd\_set
76. **if**(!curl\_multi\_fdset(cm, &fd\_read, &fd\_write, &fd\_except, &max\_fd)) {
77. fprintf(stderr, "E: curl\_multi\_fdset\n");
78. **return** EXIT\_FAILURE;
79. }
81. **if**(!curl\_multi\_timeout(cm, &curl\_timeo)) {
82. fprintf(stderr, "E: curl\_multi\_timeout\n");
83. **return** EXIT\_FAILURE;
84. }
85. **if**(curl\_timeo == -1)
86. curl\_timeo = 100;
88. //如果max\_fd返回-1，休眠一段时间后继续执行curl\_multi\_perform
89. **if**(max\_fd == -1) {
90. sleep((unsigned **int**)curl\_timeo / 1000);
91. }
92. **else** {
93. T.tv\_sec = curl\_timeo/1000;
94. T.tv\_usec = (curl\_timeo%1000)\*1000;
96. /\* 执行监听，当文件描述符状态发生改变的时候返回
97. \* 返回0，程序调用curl\_multi\_perform通知curl执行相应操作
98. \* 返回-1，表示select错误
99. \*/
100. **if**(0 > select(max\_fd+1, &fd\_read, &fd\_write, &fd\_except, &T)) {
101. fprintf(stderr, "E: select(%i,,,,%li): %i: %s\n",
102. max\_fd+1, curl\_timeo, errno, strerror(errno));
103. **return** EXIT\_FAILURE;
104. }
105. }
106. }
108. **while**((msg = curl\_multi\_info\_read(cm, &msgs\_left))) {
109. **if**(msg->msg == CURLMSG\_DONE) {
110. **char** \*url;
111. CURL \*e = msg->easy\_handle;
112. curl\_easy\_getinfo(msg->easy\_handle, CURLINFO\_PRIVATE, &url);
113. fprintf(stderr, "R: %d - %s <%s>\n",
114. msg->data.result, curl\_easy\_strerror(msg->data.result), url);
115. /\*当一个easy handler传输完成，此easy handler仍然仍然停留在multi stack中,
116. 调用curl\_multi\_remove\_handle将其从multi stack中移除,然后调用curl\_easy\_cleanup将其关闭\*/
117. curl\_multi\_remove\_handle(cm, e);
118. curl\_easy\_cleanup(e);
119. }
120. **else** {
121. fprintf(stderr, "E: CURLMsg (%d)\n", msg->msg);
122. }
123. }
124. }**while**(still\_running);
126. //当multi stack中的所有传输都完成时，调用 curl\_multi\_cleanup关闭multi handler
127. curl\_multi\_cleanup(cm);
128. curl\_global\_cleanup();
130. **return** EXIT\_SUCCESS;
131. }

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/lijinqi1987/article/details/53925835