<https://blog.csdn.net/luotuo44/article/details/38334979>

**转载请注明出处:**[**http://blog.csdn.net/luotuo44/article/details/38334979**](http://blog.csdn.net/luotuo44/article/details/38334979)

        Libevent的内存分配函数还是比较简单的，并没有定义内存池之类的东西。如同[前一篇博客](http://blog.csdn.net/luotuo44/article/details/38317797)那样，给予Libevent库的使用者充分的设置权(定制)，即可以设置用户(Libevent库的使用者)自己的内存分配函数。至于怎么分配，主动权在于用户。但在设置(定制)的时候要注意一些地方，下面会说到。

        首先，如果要定制自己的内存分配函数，就得在一开始配置编译Libevent库是，不能加入--disable-malloc-replacement选项。默认情况下，是没有这个选项的。如果加入了这个选项，那么将会在生成的event-config.h中，定义\_EVENT\_DISABLE\_MM\_REPLACEMENT这个宏。关于event-config.h文件，[可以参考博文](http://blog.csdn.net/luotuo44/article/details/38300965)。下面是Libevent内存分配函数的声明(在mm-internal.h文件)：

**[cpp]** [view plain](https://blog.csdn.net/luotuo44/article/details/38334979) [copy](https://blog.csdn.net/luotuo44/article/details/38334979)

1. //mm-internal.h文件
2. #ifndef \_EVENT\_DISABLE\_MM\_REPLACEMENT
3. **void** \*event\_mm\_malloc\_(**size\_t** sz);
4. **void** \*event\_mm\_calloc\_(**size\_t** count, **size\_t** size);
5. **char** \*event\_mm\_strdup\_(**const** **char** \*s);
6. **void** \*event\_mm\_realloc\_(**void** \*p, **size\_t** sz);
7. **void** event\_mm\_free\_(**void** \*p);
8. #define mm\_malloc(sz) event\_mm\_malloc\_(sz)
9. #define mm\_calloc(count, size) event\_mm\_calloc\_((count), (size))
10. #define mm\_strdup(s) event\_mm\_strdup\_(s)
11. #define mm\_realloc(p, sz) event\_mm\_realloc\_((p), (sz))
12. #define mm\_free(p) event\_mm\_free\_(p)
13. #else
14. #define mm\_malloc(sz) malloc(sz)
15. #define mm\_calloc(n, sz) calloc((n), (sz))
16. #define mm\_strdup(s) strdup(s)
17. #define mm\_realloc(p, sz) realloc((p), (sz))
18. #define mm\_free(p) free(p)
19. #endif

        这些内存分配函数是给Libevent使用的，而非用户(从这些接口声明在mm-internal.h文件中就可以看到这一点)。Libevent的其他函数要申请内存就调用mm\_malloc之类的宏定义。如果一开始在配置的时候(event-config.h)就禁止用户定制自己的内存分配函数，那么就把这些宏定义为C语言标准内存分配函数。

        当然，即使没有禁止，如果用户没有定制自己的内存分配函数，最终还是调用C语言的标准内存分配函数。这一点在event\_mm\_xxxx这些函数的实现上可以看到。

        这些函数的实现是在event.c文件中的。定制功能的实现原理和[前一篇博客](http://blog.csdn.net/luotuo44/article/details/38317797)中说到的定制实现原理是一样的。如下：

**[cpp]** [view plain](https://blog.csdn.net/luotuo44/article/details/38334979) [copy](https://blog.csdn.net/luotuo44/article/details/38334979)

1. #ifndef \_EVENT\_DISABLE\_MM\_REPLACEMENT
2. **static** **void** \*(\*\_mm\_malloc\_fn)(**size\_t** sz) = NULL;
3. **static** **void** \*(\*\_mm\_realloc\_fn)(**void** \*p, **size\_t** sz) = NULL;
4. **static** **void** (\*\_mm\_free\_fn)(**void** \*p) = NULL;
6. **void**
7. event\_set\_mem\_functions(**void** \*(\*malloc\_fn)(**size\_t** sz),
8. **void** \*(\*realloc\_fn)(**void** \*ptr, **size\_t** sz),
9. **void** (\*free\_fn)(**void** \*ptr))
10. {
11. \_mm\_malloc\_fn = malloc\_fn;
12. \_mm\_realloc\_fn = realloc\_fn;
13. \_mm\_free\_fn = free\_fn;
14. }

        用户就是通过调用event\_set\_mem\_functions函数来定制自己的内存分配函数。虽然这个函数不做任何的检查，但还是有一点要注意。这个三个指针，要么全设为NULL(恢复默认状态)，要么全部都非NULL。原因后面会说到。

        这些内存分配函数的实现是相当简单。看看event\_mm\_malloc\_

**[cpp]** [view plain](https://blog.csdn.net/luotuo44/article/details/38334979) [copy](https://blog.csdn.net/luotuo44/article/details/38334979)

1. **void** \*
2. event\_mm\_malloc\_(**size\_t** sz)
3. {
4. **if** (\_mm\_malloc\_fn)
5. **return** \_mm\_malloc\_fn(sz);
6. **else**
7. **return** malloc(sz);
8. }

        如果用户定制了内存分配函数(\_mm\_malloc\_fn不为NULL)，那么就直接调用用户定制的内存分配函数。否则使用C语言标准库提供的。其他几个内存分配函数也是这样实现的。这里就不贴代码了。

        定制自己的内存分配函数需要注意的一些地方：

* 替换内存管理函数影响libevent 随后的所有分配、调整大小和释放内存操作。所以必须保证在调用任何其他libevent函数之前进行定制。否则，Libevent可能用定制的free函数释放C语言 库的malloc函数分配的内存
* malloc和realloc函数返回的内存块应该具有和C库返回的内存块一样的地址对齐
* realloc函数应该正确处理realloc(NULL, sz)（也就是当作malloc(sz)处理）
* realloc函数应该正确处理realloc(ptr, 0)（也就是当作free(ptr)处理）
* 如果在多个线程中使用libevent，替代的内存管理函数需要是线程安全的
* 如果要释放由Libevent函数分配的内存，并且已经定制了malloc和realloc函数，那么就应该使用定制的free函数释放。否则将会C语言标准库的free函数释放定制内存分配函数分配的内存，这将发生错误。所以三者要么全部不定制，要么全部定制。

参考：

<http://www.wangafu.net/~nickm/libevent-book/Ref1_libsetup.html>

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/luotuo44/article/details/38334979