基本概念

　　Libevent是基于 Reactor 模式的网络库，在 Reactor 模式中，通常都有一个事件循环(Event Loop)，在 Libevent 中，这个事件循环就是event\_base结构体：

|  |  |
| --- | --- |
| 1  2  3 | struct event\_base \*event\_base\_new(void); // 创建事件循环  void event\_base\_free(struct event\_base \*base); // 销毁事件循环  int event\_base\_dispatch(struct event\_base \*base); // 运行事件循环 |

　　通常来说，事件循环主要有两个作用：

* 用来管理事件，比如说添加我们感兴趣的事件，修改事件或删除事件。
* 用来轮询它管理的所有事件，如果发现有事件活跃 (avtive)，就调用相应的回调函数去处理事件。
* Libevent 使用event结构体来代表事件，可以使用event\_new()创建一个事件：

|  |  |
| --- | --- |
| 1  2  3  4  5 | struct event \*event\_new(struct event\_base \*base, // 事件循环  evutil\_socket\_t fd, // 文件描述符  short what, // 事件类型  event\_callback\_fn cb, // 回调函数  void \*arg); // 传递给回调函数的参数 |

* 创建一个事件之后，要怎么把它加入到事件循环呢？可以使用event\_add()函数：

|  |  |
| --- | --- |
| 1  2 | int event\_add(struct event \*ev, // 事件  const struct timeval \*tv); // 超时时间 |

* 默认情况下，当一个事件变得活跃时，Libevent 会执行这个事件的回调函数，但同时也会将这个事件从事件循环中移除
* 那要怎么样才能让事件不被移除呢？当创建事件时，在事件类型加上EV\_PERSIST就可以

## TCP Server

　　Libevent 使用evconnlistener结构来表示 TCP Server，创建 TCP Server 的做法很简单：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | struct evconnlistener \*evconnlistener\_new\_bind(  struct event\_base \*base, // 事件循环  evconnlistener\_cb cb, // 回调函数，当 accept() 成功时会被调用  void \*arg, // 传递给回调函数的参数  unsigned flags, // 选项  int backlog, // tcp backlog 参数  const struct sockaddr \*sa, // 地址  int socklen  );  void evconnlistener\_free(struct evconnlistener \*lev); |

　　调用evconnlistener\_new\_bind()函数之后，listening socket 会自动被设置成非阻塞的。我们还通过flags参数设置一些有用的选项，例如：

* LEV\_OPT\_CLOSE\_ON\_FREE表示当调用evconnlistener\_free()时，相应的 listening socket 也会被close()掉。
* LEV\_OPT\_REUSEABLE表示会自动对 listening socket 设置SO\_REUSEADDR这个 TCP 选项。