**创建默认的event\_base**

event\_base\_new()函数分配并且返回一个新的具有默认设置的 event\_base。如果发生错误,则返回 NULL。

struct event\_base \*event\_base\_new(void);

大多数程序使用这个函数就够了。

**创建复杂的event\_base**

要创建复杂的event\_base就需要使用 event\_config。

event\_config 是一个容纳 event\_base 配置信息的不透明结构体。

如下函数对event\_config进行操作：

需要 event\_base 时,将 event\_config 传递给event\_base\_new\_with\_config ()。

**创建接口event\_config**

struct event\_config \*event\_config\_new(void);

**由event\_config生成event\_base**

struct event\_base \*

event\_base\_new\_with\_config(const struct event\_config \*cfg);

**释放event\_config**

void event\_config\_free(struct event\_config \*cfg);

**event\_config\_avoid\_method**

通过名字让 event\_base 避免使用指定方法的后端

int event\_config\_avoid\_method(struct event\_config \*cfg, const char \*method);

**event\_config\_require\_feature**

指定event\_base必须的特征

enum event\_method\_feature {

EV\_FEATURE\_ET = 0x01,

EV\_FEATURE\_O1 = 0x02,

EV\_FEATURE\_FDS = 0x04,

};

int event\_config\_require\_features(struct event\_config \*cfg,

enum event\_method\_feature feature);

**event\_config\_set\_flag**

设置一个或多个event\_base标记

enum event\_base\_config\_flag {

EVENT\_BASE\_FLAG\_NOLOCK = 0x01,

EVENT\_BASE\_FLAG\_IGNORE\_ENV = 0x02,

EVENT\_BASE\_FLAG\_STARTUP\_IOCP = 0x04,

EVENT\_BASE\_FLAG\_NO\_CACHE\_TIME = 0x08,

EVENT\_BASE\_FLAG\_EPOLL\_USE\_CHANGELIST = 0x10,

EVENT\_BASE\_FLAG\_PRECISE\_TIMER = 0x20

};

int event\_config\_set\_flag(struct event\_config \*cfg,

enum event\_base\_config\_flag flag);

EV\_FEATURE\_ET:要求支持边沿触发的后端

EV\_FEATURE\_O1:要求添加、删除单个事件,或者确定哪个事件激活的操作是 O(1)复杂度的后端

EV\_FEATURE\_FDS:要求支持任意文件描述符,而不仅仅是套接字的后端

event\_config\_set\_flag()可识别的选项值有:

EVENT\_BASE\_FLAG\_NOLOCK :不要为 event\_base 分配锁。设置这个选项可以 为 event\_base 节省一点用于锁定和解锁的时间,但是让在多个线程中访问 event\_base 成为不安全的。

EVENTBASE\_FLAG\_IGNORE\_ENV :选择使用的后端时,不要检测 EVENT\* 环境 变量。使用这个标志需要三思:这会让用户更难调试你的程序与 libevent 的交互。

EVENT\_BASE\_FLAG\_STARTUP\_IOCP:仅用于 Windows,让 libevent 在启动时就 启用任何必需的 IOCP 分发逻辑,而不是按需启用。

EVENT\_BASE\_FLAG\_NO\_CACHE\_TIME :不是在事件循环每次准备执行超时回调时 检测当前时间,而是在每次超时回调后进行检测。注意:这会消耗更多的 CPU时间。

EVENT\_BASE\_FLAG\_EPOLL\_USE\_CHANGELIST :告诉 libevent ,如果决定使 用 epoll 后端,可以安全地使用更快的基于 changelist 的后端。epoll-changelist 后端可以 在后端的分发函数调用之间,同样的 fd 多次修改其状态的情况下,避免不必要的系统 调用。但是如果传递任何使用 dup()或者其变体克隆的 fd 给 libevent,epoll-changelist 后端会触发一个内核 bug,导致不正确的结果。在不使用 epoll 后端的情况下,这个标 志是没有效果的。也可以通过设置

EVENT\_EPOLL\_USE\_CHANGELIST 环境变量来 打开 epoll-changelist 选项。

上述操作 event\_config 的函数都在成功时返回0,失败时返回-1。