在Libev的源码中，用到了一种用C实现类似C++中继承的技巧，主要是用宏和结构体实现。

在Libev中，最关键的数据结构就是各种监视器，比如IO监视器，信号监视器等等。这些监视器的多数成员都是一样的，只有少部分成员为各自独有。这就非常类似于C++中继承的使用场景了。废话少说，代码如下（略有改动，某些宏做了展开）：

# define EV\_CB\_DECLARE(type) void (\*cb)(struct ev\_loop \*loop, struct type \*w, int revents);

#define EV\_WATCHER(type) \

int active; /\* private \*/ \

int pending; /\* private \*/ \

int priority; /\* private \*/ \

void \*data; /\* rw \*/ \

EV\_CB\_DECLARE (type) /\* private \*/

#define EV\_WATCHER\_LIST(type) \

EV\_WATCHER (type) \

struct ev\_watcher\_list \*next; /\* private \*/

**typedef** struct ev\_watcher

**{**

EV\_WATCHER **(**ev\_watcher**)**

**}** ev\_watcher**;**

**typedef** struct ev\_watcher\_list

**{**

EV\_WATCHER\_LIST **(**ev\_watcher\_list**)**

**}** ev\_watcher\_list**;**

**typedef** struct ev\_io

**{**

EV\_WATCHER\_LIST **(**ev\_io**)**

int fd**;** /\* ro \*/

int events**;** /\* ro \*/

**}** ev\_io**;**

**typedef** struct ev\_signal

**{**

EV\_WATCHER\_LIST **(**ev\_signal**)**

int signum**;** /\* ro \*/

**}** ev\_signal**;**

ev\_watcher是所有结构的“基类”，宏EV\_WATCHER定义了它的所有成员。像IO监视器、信号监视器等是以链表的形式进行组织的，所以，在ev\_watcher基类的基础上，定义了ev\_watcher的子类ev\_watcher\_list，宏EV\_WATCHER\_LIST就定义了该基类的所有成员。

ev\_io和ev\_signal相当于ev\_watcher\_list的子类，它们的前5个成员于ev\_watcher相同，前6个成员与ev\_watcher\_list相同。

上面的代码可能还不是太清楚，下面将上面代码中的所有宏都展开，代码如下：

# define EV\_CB\_DECLARE(type) void (\*cb)(struct ev\_loop \*loop, struct type \*w, int revents);

#define EV\_WATCHER(type) \

int active; /\* private \*/ \

int pending; /\* private \*/ \

int priority; /\* private \*/ \

void \*data; /\* rw \*/ \

EV\_CB\_DECLARE (type) /\* private \*/

#define EV\_WATCHER\_LIST(type) \

int active; /\* private \*/ \

int pending; /\* private \*/ \

int priority; /\* private \*/ \

void \*data; /\* rw \*/ \

EV\_CB\_DECLARE (type) /\* private \*/ \

struct ev\_watcher\_list \*next; /\* private \*/

**typedef** struct ev\_watcher

**{**

int active**;**

int pending**;**

int priority**;**

void **\***data**;**

EV\_CB\_DECLARE **(**ev\_watcher**)**

**}** ev\_watcher**;**

**typedef** struct ev\_watcher\_list

**{**

int active**;**

int pending**;**

int priority**;**

void **\***data**;**

EV\_CB\_DECLARE **(**ev\_watcher\_list**)**

struct ev\_watcher\_list **\***next**;**

**}** ev\_watcher\_list**;**

**typedef** struct ev\_io

**{**

int active**;**

int pending**;**

int priority**;**

void **\***data**;**

EV\_CB\_DECLARE **(**ev\_io**)**

struct ev\_watcher\_list **\***next**;**

int fd**;** /\* ro \*/

int events**;** /\* ro \*/

**}** ev\_io**;**

**typedef** struct ev\_signal

**{**

int active**;**

int pending**;**

int priority**;**

void **\***data**;**

EV\_CB\_DECLARE **(**ev\_signal**)**

struct ev\_watcher\_list **\***next**;**

int signum**;**

**}** ev\_signal**;**

下面是Libev中如何使用这些结构体的代码：

void ev\_start **(**struct ev\_loop **\***loop**,** ev\_watcher**\*** w**,** int active**)**

**{**

**...**

w**->**active **=** active**;**

**...**

**}**

void wlist\_add **(**ev\_watcher\_list **\*\***head**,** ev\_watcher\_list **\***elem**)**

**{**

elem**->**next **=** **\***head**;**

**\***head **=** elem**;**

**}**

void ev\_io\_start **(**struct ev\_loop **\***loop**,** ev\_io **\***w**)**

**{**

**...**

ev\_start **(**loop**,** **(**ev\_watcher**\*)**w**,** 1**);**

**...**

wlist\_add **(&**anfds**[**fd**].**head**,** **(**ev\_watcher\_list **\*)**w**);**

**}**

在ev\_io\_start函数中，w是指向ev\_io结构的指针，使用ev\_start函数设置其成员active时，将其强制转换成基类ev\_watcher，在将其添加进链表时，又将其强制转化为ev\_watcher\_list类型。

下面的代码是模拟上面的方法，写出的例子代码：

**typedef** struct

**{**

char **\***color**;**

int weight**;**

**}**fruit**;**

**typedef** struct fruitlist

**{**

char **\***color**;**

int weight**;**

struct fruitlist **\***next**;**

**}**fruitlist**;**

**typedef** struct

**{**

char **\***color**;**

int weight**;**

struct fruitlist **\***next**;**

char **\***taste**;**

**}**apple**;**

void setcolorweight**(**fruit **\***f**,** char **\***color**,** int weight**)**

**{**

f**->**color **=** color**;**

f**->**weight **=** weight**;**

**}**

void putinlist**(**fruitlist **\*\***head**,** fruitlist **\***ele**)**

**{**

ele**->**next **=** **\***head**;**

**\***head **=** ele**;**

**}**

void testinherit**()**

**{**

fruitlist **\***head **=** **NULL;**

apple ap1**;**

setcolorweight**((**fruit **\*)&**ap1**,** "red"**,** 2**);**

ap1**.**taste **=** "sweet"**;**

putinlist**(&**head**,** **(**fruitlist **\*)&**ap1**);**

apple ap2**;**

setcolorweight**((**fruit **\*)&**ap2**,** "yellow"**,** 1**);**

ap2**.**taste **=** "sour"**;**

putinlist**(&**head**,** **(**fruitlist **\*)&**ap2**);**

fruitlist **\***p **=** head**;**

**while(**p **!=** **NULL)**

**{**

printf**(**"color is %s, weight is %d\n"**,** p**->**color**,** p**->**weight**);**

p **=** **(**fruitlist **\*)(**p**->**next**);**

**}**

**}**

color is yellow, weight is 1

color is red, weight is 2