# 1 Thread模块

提供线程接口，用作让schedule模块继承，并完成线程的操作。一般Schedule模块继承于Thread模块，实现其虚函数也是线程的主函数work，定义run和end接口，run中调用activate启动线程，在end调用wait销毁线程；

主要成员变量：

pthread\_t \*m\_handle; //线程句柄

int m\_nThr; //线程计数，可以启动多个线程

主要成员函数：

int activate(int n=1)：启动线程pthread\_create，默认启动第一个，线程名称是\_work\_T，线程传入的参数是继承Thread的变量及线程的索引i创建的ThreadData\_t变量，这样就把子类传进去了，创建多个线程时，在中间会有个延时；

int wait()：依次等待线程毁灭pthread\_join；

virtual int work(int e) ：虚函数，让你类实现，也是实际执行的线程函数；

static void\* \_work\_T(void\* p)：静态函数，在activate中被调用，实际执行的是子类的work；

int get\_work\_i(); //定义多条线程从0开始编号，0表示第1条，返回的就是这个值

## 1.1 Scheduler

继承Thread，主要现接口：

int run(); //启动线程，实际上会调用父类的activate，还会set线程run的flag；

void end(); //清空线程run的flag，调用父类的wait等待线程退出

virtual int work(int e); //覆盖父类的方法，是个while，实际创建的线程



# 2 PTLStream模块

继承于SerialStream，完成数据的各种操作。

SerialStream的主重要成员变量

char m\_endian; //大小端

char\* m\_buf; //实际数据的存储

int m\_size; //buffer的大小

int m\_state; //

int m\_rpos; //读位置

int m\_wpos; //写位置

bool m\_mynew; //buffer是否是自己new的

SerialStream的成员函数主要是操作读写指针的，一看就懂

Note：

1. PTLStream的fitsize32用于给指定位置加上32位包大小。

2.在tracker中常与MemBlock结合使用，MemBlock新建buffer，让PTLStream使用；

# 3 Speedometer模块

速度统计模块，定义的是个模板类，template<typename T,unsigned int LEN=60>，两个参数，第二参数默认为60。

重要成员变量

T m\_total; //总的数据大小

T m\_last; //记录某一s内数据的大小

T m\_data[LEN]; //记录60s内每秒内传输数据大小

unsigned int m\_cursor; //当前的秒数

unsigned int m\_sec\_amount; //统计的秒数

unsigned int m\_last\_tick;

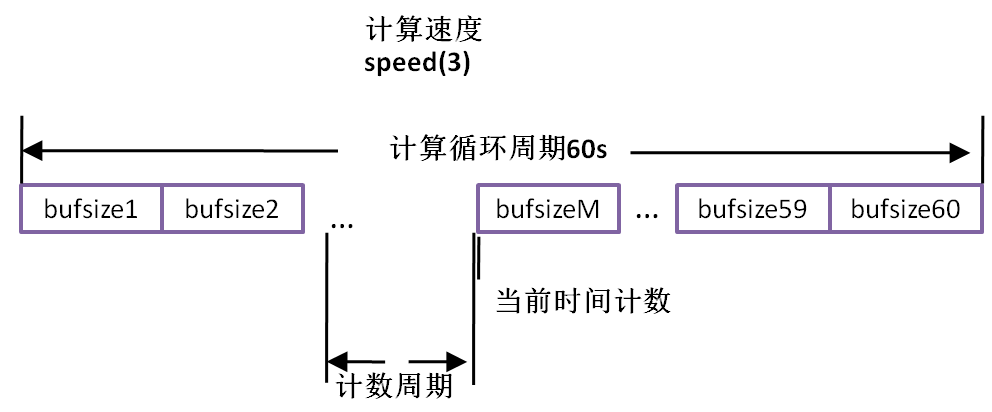
unsigned int m\_limit\_speed; //限速，即每秒发送的数据大小；

重要成员函数

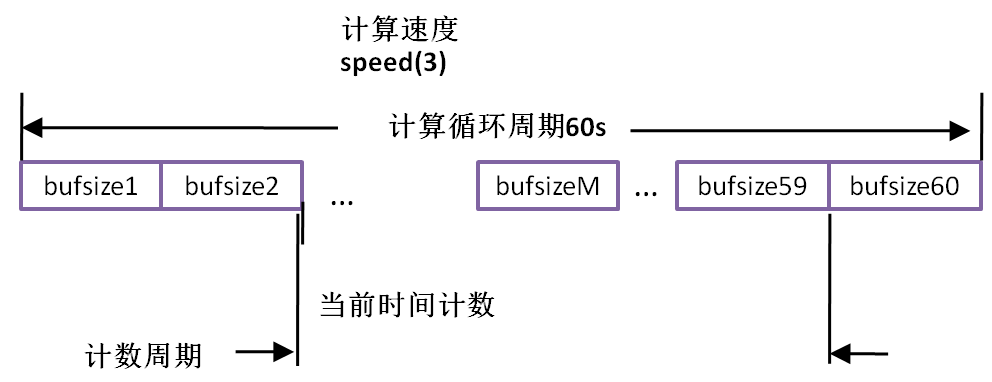
on\_second：切换当前秒的统计；

get\_speed：获取速度，传入统计的范围秒数，从当前秒数开始向后和向前的速度统计，若统计的秒数为负，则计算的全局的总平均速度；

limit\_recv：计算限速度下本秒应该接收的数据大小；



speed=(bufsizeM + bufsizeM-1 +bufsizeM-2)/3



speed=(bufsize2 + bufsize1 +bufsize60)/3

# 4.Speaker模块

回调函数的重要数据结构，模板类template<typename Listener>，主要用于调用注册的回调函数（继承于Listener）

成员变量

int \_size;

Listener \*\*\_listeners; //已注册的回调函数实体类

Listener \*\*\_tmp\_listeners; //作临时变量用

int \_cursor; //已注册的size

主要成员函数

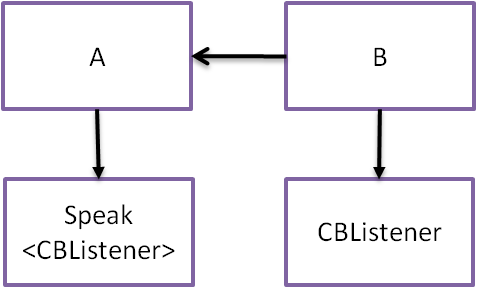
add\_listener：添加回调函数的实体；

remove\_listener：移除；

find：查找某个listener；

Note：

1.其中的各fire函数定义的是模板函数，会调用listener中定义的各种on函数；



2.常见用法，B中有A的实体，A继承于Speak<CBListener>，B继承于CBListener，B实现了CBListener中定义的各种on函数，在B中，可以调用A的add\_listener，这样在A中就可以通过调用fire函数调用A中相应的on函数，就实现的回调；

# 5.List模块

链表参考的是linux链表设计原则，定义的链表头如下，把链表头包括在需要串在一起的结构中就能实现

typedef struct rlist\_head

{

struct rlist\_head \*next,\*prev;

}rlist\_head\_t;

\_\_rlist\_join：将两个结点连接起来；

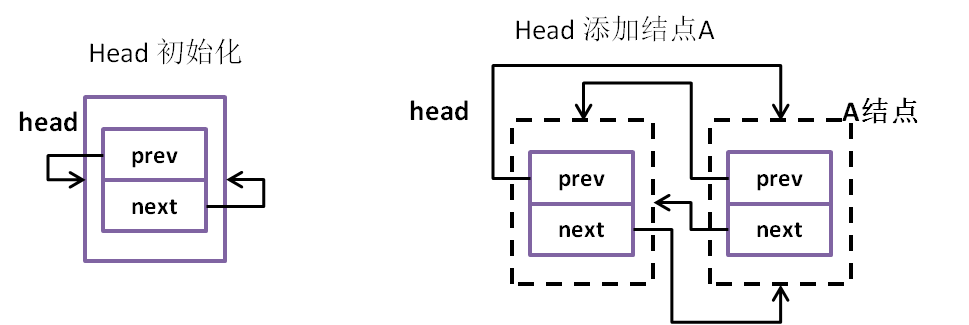
rlist\_init：初始化结点头；

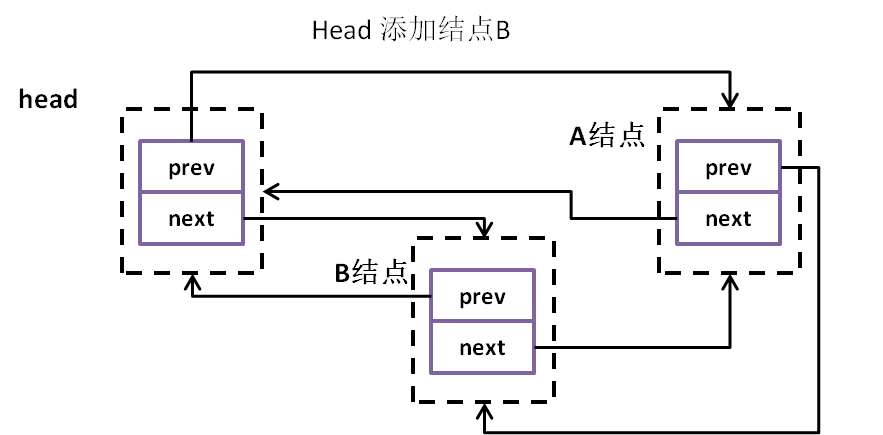
rlist\_empty：结点清空；

rlist\_add\_head：从头端添加结点；

rlist\_add\_tail：从未雄姿英发添加结点；

rlist\_del：删除结点；





#define rlist\_entry(ptr,type,member) \

((type\*)((char\*)(ptr)-(char\*)&((type\*)8)->member+8))

# 6.Http模块

http模块，httpserver响应http客户端的请求，接收到一个http请求后会创建一个httpclient处理响应事宜，支持多线程，如使用多线程的会，收到请求后会创建一个线程专程负责；

httpserver主要成员函数

SetHandleReqFun(FUN\_HANDLE\_HTTP\_REQ\_PTR fun)：设置http的响应函数，这个函数会在解析http头后调用处理；

open：创建http通道；

\_startT(void \*p)：httpserver的主要处理部分，是静态成员函数；

httpclient是服务器端专门处理客户端的响应设计的结构，其最主要的处理是调用svc处理

handle\_request：；

handle\_head：head解析，把head，cgi与parm解析出，解析完会调用传入的回调函数m\_fun\_handle\_http\_req处理；

# 7.Timer模块

## 7.1定义接口TimerHandler

virtual void on\_timer(int e)：对外接口，一个Handler可以向timer注册好几次，因其传入的e不同，实际调用时会调用不同的部分；

## 7.2延时队列DelayQueue

定义的protected的接口如下：

void add\_node(TNode\_t\* node); //增加结点

void del\_node(TNode\_t\* node); //删除结点

TNode\_t\* get\_zero\_delay(); //获取时间到了的结点

void synchronize()

成员变量：

TNode\_t m\_head; //结点头

ULONGLONG m\_last\_tick;

int m\_size; //结点的计数

其中的结点的结构tagTNode也即TNode\_t如下：

TimerHandler \*h; //结点到时会触发的函数

int e; //eventid

ULONGLONG us; //timeout

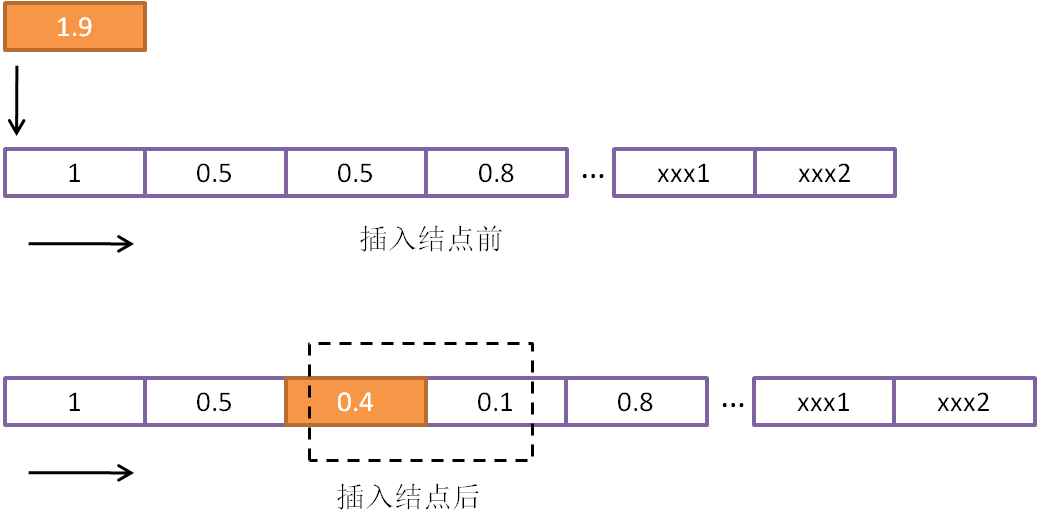
//DWORD next\_us;//下一次执行时间

ULONGLONG remain\_us; //剩余等待时间

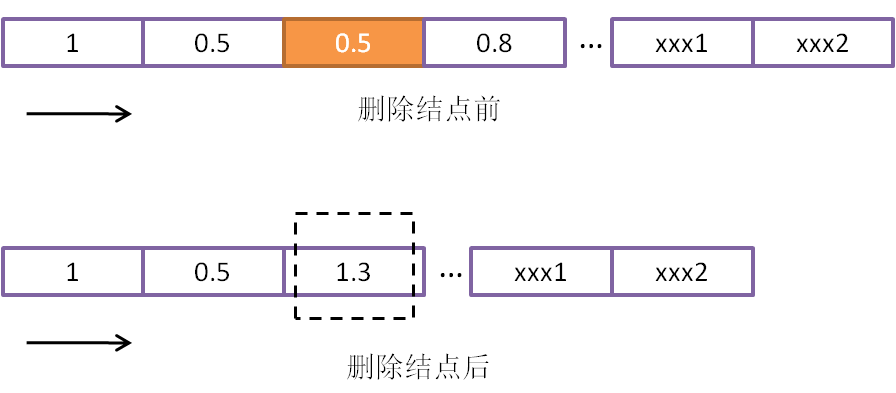
tagTNode \*next;

tagTNode \*pre;

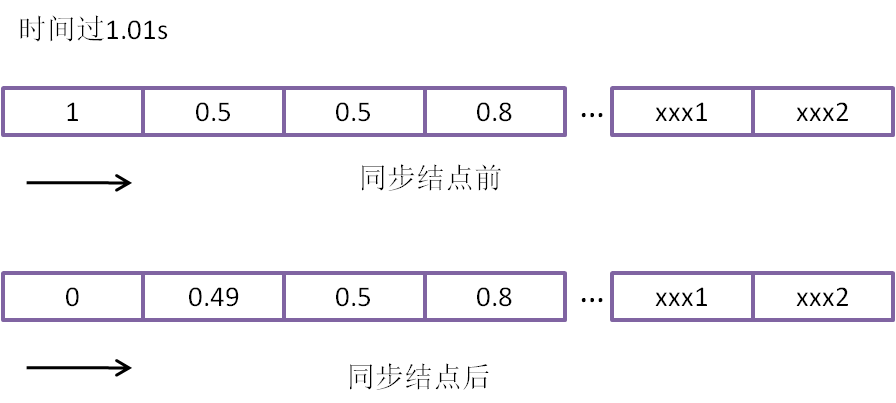
延时队列中结点的remain时间是与前时间的差，如下面第1个结点是1s后触发，第2个结点是1.5s后触发，即第一个结点触发后0.5s后触发，添加一个结点add\_node的过程如图所示。



结点删除del\_node的过程如下图，被删除结点的后一个结点的remain时间会加上被删除结点的时间



结点的同步synchronize，主要是计算这次与上次的时间差，并修改结点的remain，一般会只有修改一个结点的值。下图中，同步后出现remain时间为0，这样就可以把这个结点取出；



## 7.3 Timer模块

Timer模块继承于DelayQueue，

主要接口如下：

oid handle\_root(); //时间模块的主处理部分，先调用同步synchronize，再取出remain时间为0的结点，执行其on\_timer模块，执行完会把结点从中删除再添加，这样就实际定时任务；

int register\_timer(TimerHandler \*h,int e,DWORD ms);//向时间模块注册一个结点，时间单位是ms，并把handler注册过去，这样到时间会自动调用其函数，注册时的结点的实际存储空间是从m\_nl数组中宴请的，

int register\_utimer(TimerHandler \*h,int e,ULONGLONG us);

int unregister\_timer(TimerHandler \*h,int e); //取消时间结点

int unregister\_all(TimerHandler \*h)

int find(TimerHandler \*h,int e); //查找同个handler及两样事件e的结点，是从数组中查找的；

int allot(); //从队列中申请结点

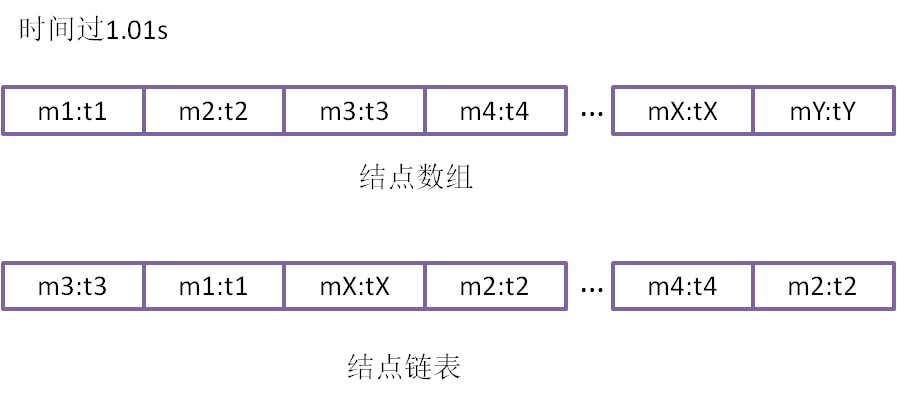
void free(int i)；//取消某个结点

主要成员变量

TNode\_t \*m\_nl; //结点数据

int m\_nl\_size; //允许的结点个数

int m\_cursor; //当前结点的个数



总体上应该是如上图，从静态数组中按序分配结点，再按remain时间放入结点的链表中，结点数组才是结点的实体；