每个worker线程通过管道方式与其它线程（主要是主线程）进行通信，调用pipe函数，产生两个fd，一个是管道写入fd，一个是管道读取fd。worker线程把管道读取fd加到自己的event\_base，监听管道读取fd的可读事件，即当主线程往某个线程的管道写入fd写数据时，触发事件。

主线程监听到有一个连接到达时，accept连接，产生一个client fd，然后选择一个worker线程，把这个client fd包装成一个CQ\_ITEM对象（该结构体下面再详细讲，这个对象实质是起主线程与worker线程之间通信媒介的作用，主线程把client fd丢给worker线程往往不止“client fd”这一个参数，还有别的参数，所以这个CQ\_ITEM相当于一个“参数对象”，把参数都包装在里面），然后压到worker线程的CQ\_ITEM队列里面去（每个worker线程有一个CQ\_ITEM队列）， 同时主线程往选中的worker线程的管道写入fd中写入一个字符“c”（触发worker线程）。

主线程往选中的worker线程的管道写入fd中写入一个字符“c”，则worker线程监听到自己的管道读取fd可读，触发事件处理，而此是的事件处理是：从自己的CQ\_ITEM队列中取出CQ\_ITEM对象（相当于收信，看看主线程给了自己什么东西），从4）可知，CQ\_ITEM对象中包含client fd，worker线程把此client fd加入到自己的event\_base，从此负责该连接的读写工作。

memcached主线程和worker线程各有自己的监听队列，故有主线程和每个worker线程都有一个独立的event\_base，事件基地。

这里是一个有向图，每个case是一个顶点，有些case通过改变conn对象的连接状态让程序在下一次循环中进入另一个case，几次循环后程序最终进入到“无出度的顶点”然后结束状态机，这里的无出度的顶点就是带设置stop=true的case分支。

在 Linux 的缓存 IO 机制中，操作系统会将 IO 的数据缓存在文件系统的页缓存（ page cache ）中，也就是说，数据会先被拷贝到操作系统内核的缓冲区中，然后才会从操作系统内核的缓冲区拷贝到应用程序的地址空间。

网络IO的本质是socket的读取，socket在linux系统被抽象为流，IO可以理解为对流的操作。刚才说了，对于一次IO访问（以read举例），数据会先被拷贝到操作系统内核的缓冲区中，然后才会从操作系统内核的缓冲区拷贝到应用程序的地址空间。所以说，当一个read操作发生时，它会经历两个阶段： 第一阶段：等待数据准备 (Waiting for the data to be ready)。

第二阶段：将数据从内核拷贝到进程中 (Copying the data from the kernel to the process)。

对于socket流而言， 第一步：通常涉及等待网络上的数据分组到达，然后被复制到内核的某个缓冲区。

第二步：把数据从内核缓冲区复制到应用进程缓冲区。

网络应用需要处理的无非就是两大类问题，网络IO，数据计算。相对于后者，网络IO的延迟，给应用带来的性能瓶颈大于后者。

为ht[1]分配空间，让字典同时持有ht[0]和ht[1]两个哈希表。

在字典中维持一个索引计数器变量rehashidx，并将它的值设置为0，表示rehash工作正式开始。

在rehash进行期间，每次对字典执行添加、删除、查找或者更新操作时，程序除了执行指定的操作以外，还会顺带将ht[0]哈希表在rehashidx索引上的所有键值对rehash到ht[1]，当rehash工作完成之后，程序将rehashidx属性的值增一。

随着字典操作的不断执行，最终在某个时间点上，ht[0]的所有键值对都会被rehash至ht[1]，这时程序将rehashidx属性的值设为-1，表示rehash操作已完成。