通信库文档

一、EventLoop

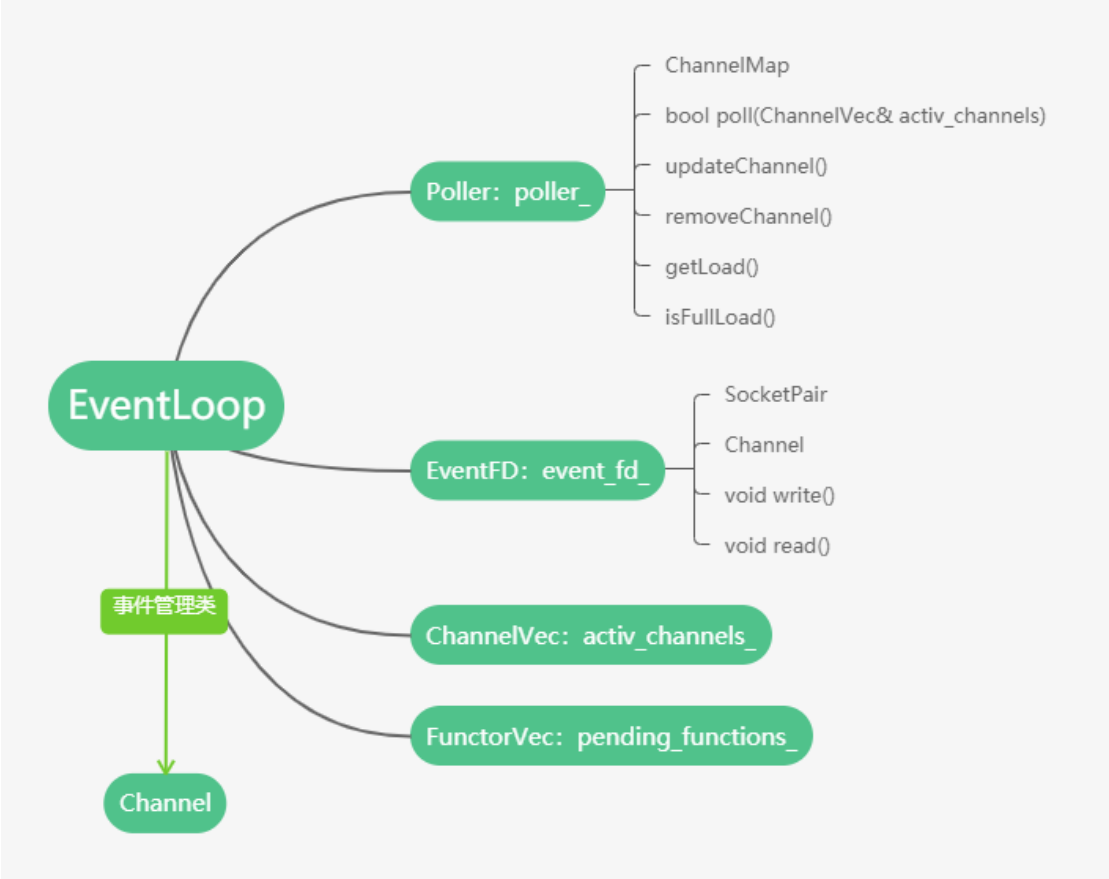

图 1-1 EventLoop 思维导图

事件循环的管理者，通过 bool loop()启动事件循环，void quit()退出事件循环。启动事件循环后调用 Poller 对象监听所有关心的 I/O 事件，当事件触发后通过 activ_channels_对象处理所有触发事件，再通过 pending_functions_对象处理所有临时函数对象。最后重新通过 Poller 对象回到监听状态。

1.1、Poller:poller_

通过 select 进行 I/O 复用，监听读、写、error 事件，并分发到 EventLoop 中。持有并管理 EventLoop 中所有的 Channel 对象。

ChannelMap

持有当前 EventLoop 中所有的 Channel 对象，其中 key 是文件描述符 fd，value 是 Channel 对象。

bool poll(ChannelVec& activ_channels)

在事件循环中，负责阻塞事件循环，当有读、写等事件发生时会被唤醒，通过 findActivChannels 将事件对应的 Channel 对象放到传入到参数 activ_channels 中。

updateChannel()

添加、删除或修改 Channel 对象所关心的事件时被调用，将改动的事件同步修改到当前 select 所监听的事件。

removeChannel()

移除一个所关心的 Channel 对象

getLoad()

获取当前 select 函数的负载情况。

isFullLoad()

判断当前 select 是否满载。

1.2、EventFD:event_fd_

持有一对已经建立的 tcp 文件描述符，通过 Channel 监听其中一个 fd 的读事件，将 Channel 对象注册到 EventLoop 中。当需要手动唤醒 Poller::poll::select 的阻塞时，向另一个 fd 写一个 byte 的数据，会触发 Channel 的读事件。从而使线程从 select 调用中唤醒。

SocketPair:pair_

创建一对已经连接的 Tcp 文件描述符

Channel:channel_

关联并监听 SocketPair 中的一个文件描述符的读事件。

void write()

向文件描述符中写一个字节，如果当前时间循环线程阻塞该函数可以唤醒当前线程。

void read()

从另一个文件描述符中读最大 1024 个字节，防止 Tcp 滑动窗口阻塞。

1.3、ChannelVec:activ_channels_

作为 Poller::poll 函数的传入参数。

poll 函数返回后，vector 中会填充有事件触发的 Channel 对象。而后 EventLoop 将会调用每个 Channel 对象的事件处理函数 Channel::handleEvent()，然后清空 activ_channels_列表。

1.4、FunctorVec:pending_functions_

vector 中存放临时需要在当前线程中调用的函数对象 Functor\Function<void(void)>。每当有函数对象通过 EventLoop 接口：void runInLoop(Functor funtor)或 void queueInLoop(const Functor& funtor)被添加到 vector 时，都有可能通过 EventFd 对象唤醒当前被 select 阻塞的线程。线程唤醒后，处理完 activ_channels_对象后，紧接将当前 vector 拷贝一份，并清空 pending_functions_对象，然后调用拷贝出的 vector 中每一个函数对象。

二、Channel

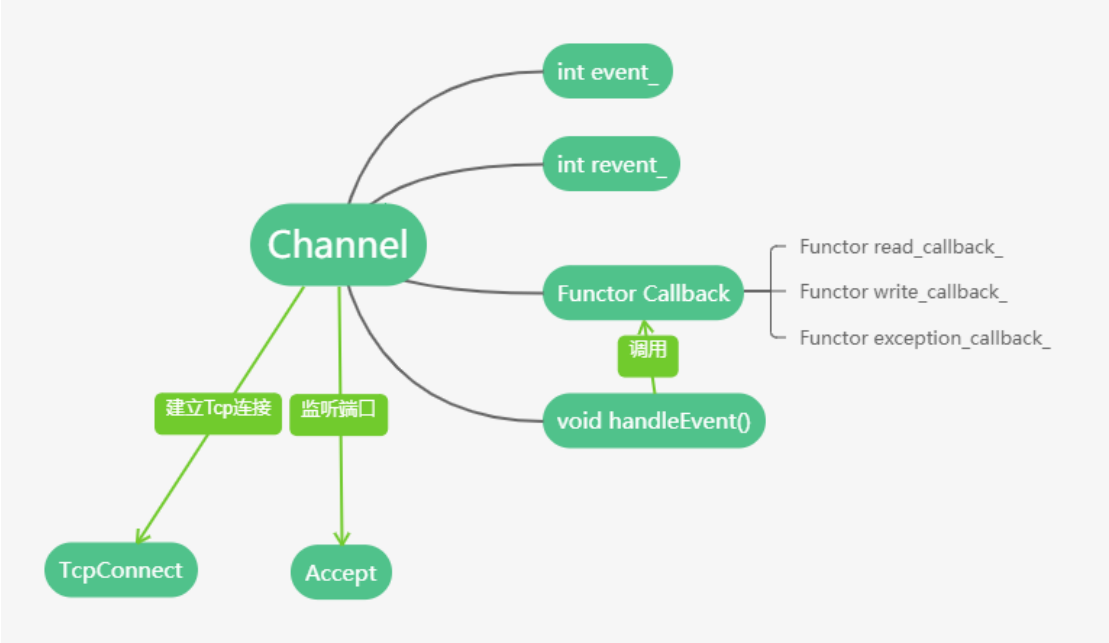

图 2-1 Channel 思维导图

Socket 文件描述符的事件监听以及处理函数的管理者，是事件循环与外部事件处理的通道。构造时与一个事件循环与 Socket 文件描述符绑定。支持三种事件：读事件、写事件、异常事件(错误处理事件)，每个事件对应一个供外部注册的回调函数。

2.1、int event_

用于存储当前所关注的事件，通过 enableReadEvent()、disableReadEvent()（函数内是与、或操作和 update（）操作）等接口进行修改。

2.2、int revent_

用于存储当前触发的事件，通过 setREvent()接口设置（由 Poller 对象调用）

2.3、Functor Callback

Functor read_callback_

触发读事件时调用的函数对象，外部通过 setReadCallBack(…)接口设置。

Functor write_callback_

触发写事件时调用的函数对象，外部通过 setWriteCallBack（…)接口设置。

Functor exception_callback_

触发异常事件时调用的函数对象，外部通过 setExceptionCallBack（…)接口设置。

2.4、void handleEvent()

事件处理方法，当前对象有关心的事件被触发时被调用（由 Poller 对象调用），通过判断 revent_的状态调用不同的 Function Callback。处理事件时会将 porcessing_event_设置为 true 结束后会将 porcessing_event_设回 false，并清空 revent_。

三、Accept

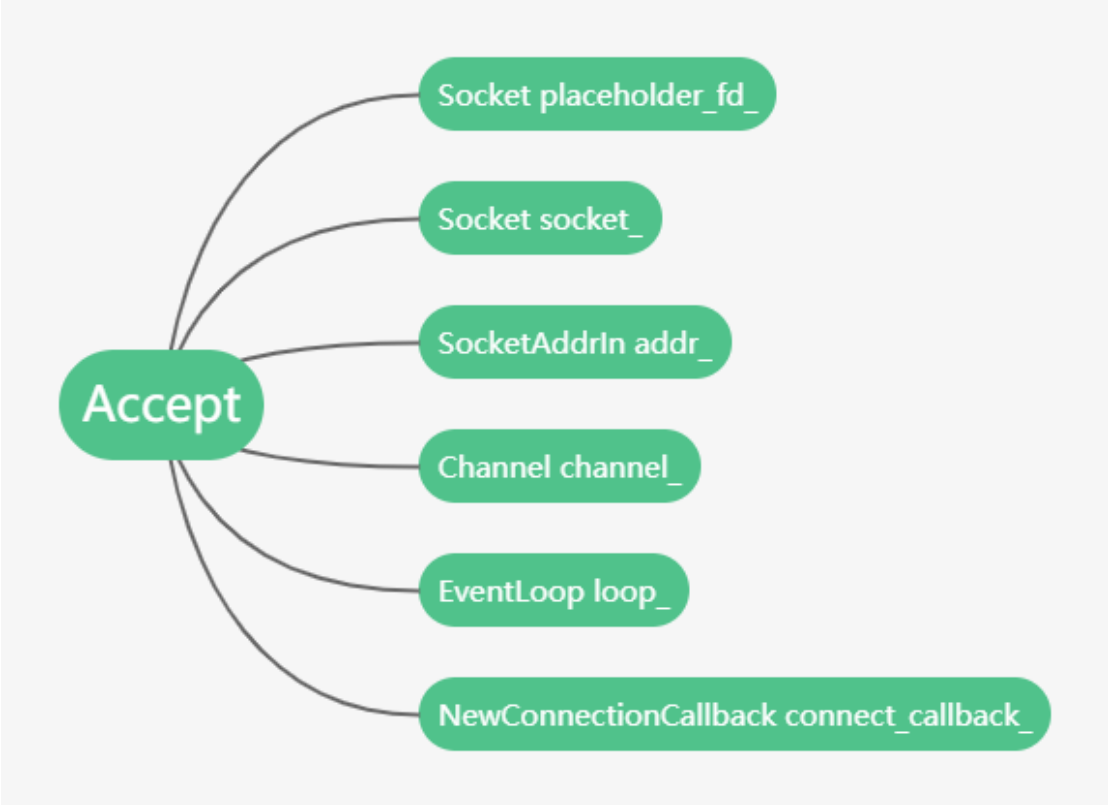

图 3-1 Accept 思维导图

Accept 用于监听端口，并处理新连接。通过 Channel 与 EventLoop 进行交互。将需要监听的文件描述符与 Channel 进行绑定，并注册读事件，当有新的连接请求时，事件循环将会调用 Accept 的 handleRead()接口对新连接进行处理。

3.1、Socket placeholder_fd_

文件描述符占位，用于防止文件描述符耗尽的情况下，可以正确关闭多余的连接。

3.2、Socket socket_

当前监听的文件描述符，在构造函数中创建，并绑定传入的地址。

3.3、SocketAddrIn addr_

当前监听的 ip 和端口号，在构造函数中传入。

3.4、Channel *channel_

用于接入事件循环，在构造函数在创建，与 socket_对象绑定。

3.5、EventLoop *loop_

所关联的事件循环，在构造函数中传入。

3.6、NewConnectionCallback connect_callback_

新连接创建后的回调函数，会在 handleRead（）接口中调用。

四、TcpConnect

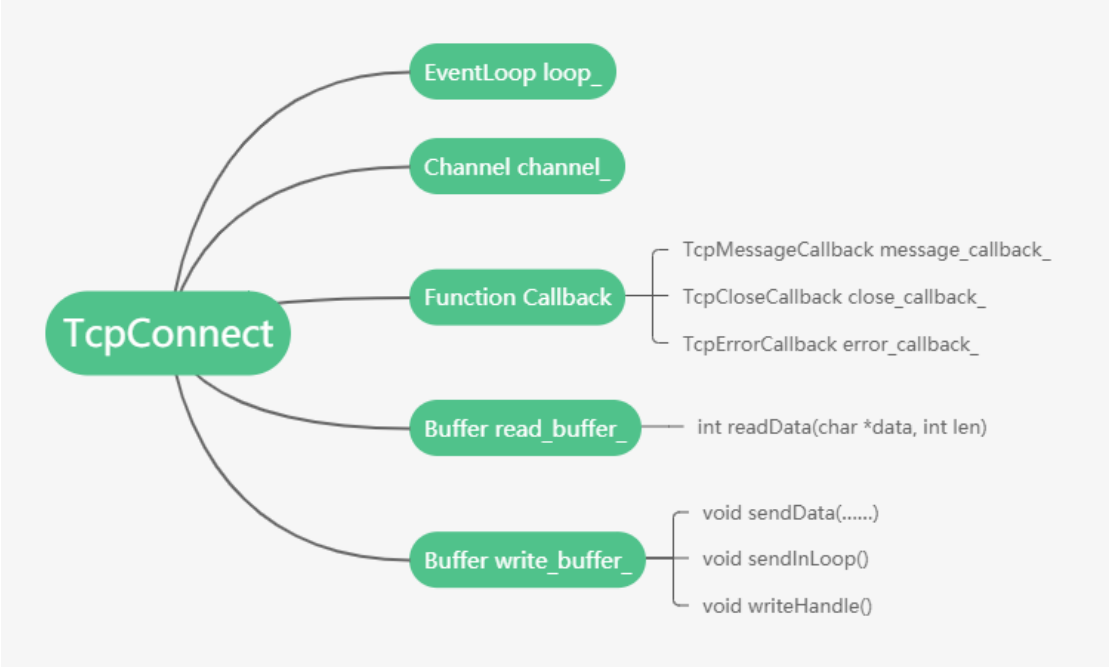

图 4-1 TcpConnect 思维导图

TcpConnect 是 tcp 连接的管理者，其生命周期等同于一次 tcp 连接，在每当有一个新连接到来时被创建，在连接断开时被销毁。用户通过它来管理每一次的读写事件，不需要再考虑发送时不完整。每一次读都是同步的，每一次发送都是添加到缓冲区再异步发送。

4.1、EventLoop loop_

构造时需要关联到一个事件循环

4.2、Channel channel_

构造时需要创建一个 Channel 对象通过事件循环监听 I/O 事件

4.3、Function Callback

TcpMessageCallback message_callback_

当事件循环接收到消息时触发回调，通过接口 setMessageCallback(const TcpMessageCallback& callback)设置回调。

TcpCloseCallback close_callback_

当连接关闭时触发回调，通过接口 void setCloseCallback(const TcpCloseCallback& callback)设置回调。

TcpErrorCallback error_callback_

当读数据或则写数据发送错误时触发回调，通过接口 void setErrorCallback(const TcpErrorCallback& callback)设置回调。

4.4、Buffer read_buffer_

当事件循环触发读事件时，会向 read_buffer_中写入数据，然后触发 message_callback_回调函数，可以在回调函数中通过 int readData(……)接口读取 buffer 中的数据。也可以在当前线程直接调用 int readData(……)接口读取 buffer 中的数据。

int readData(……)

读取 read_buffer_中的数据，线程安全函数。

4.5、Buffer write_buffer_

通过接口 void sendData(……)向 write_buffer_的尾部追加数据，在 void sendInLoop()或 void writeHandle()接口中将 buffer 中的数据真正发送出去。

void sendData(……)

向 write_buffer_的尾部追加数据，并调用 void sendInLoop()接口发送数据。线程安全函数。

void sendInLoop()

尝试发送所有 buffer 数据，如果不能一次发送完(因为滑动窗口)，则开启监听可写事件，当可写事件触发时通过 void writeHandle()接口发送剩余数据。

void writeHandle()

每当可写事件触发时被调用，尝试发送剩余的 write_buffer_中的数据，直到 buffer 数据为空时关闭监听可写事件。

五、TcpServer

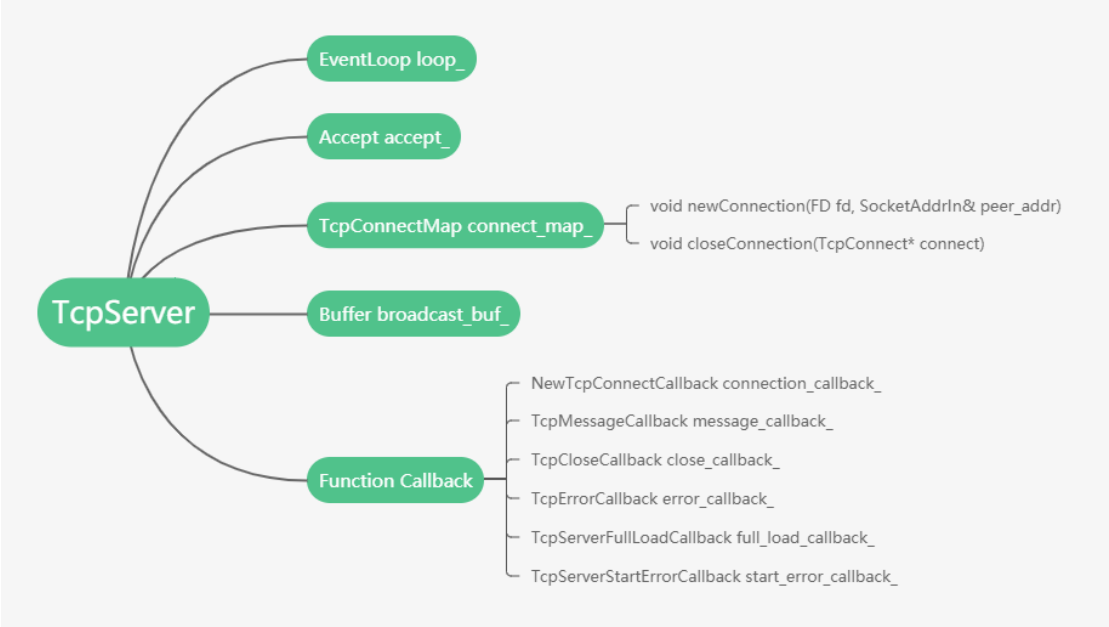

图 5-1 TcpServer 思维导图

负责 Tcp 端口监听以及管理所有连接到该端口的 Tcp 连接。TcpServer 提供 6 个回调接口，其中 TcpServerStartErrorCallback start_error_callback_是针对 TcpServer 启动失败的回调一般是因为事件循环满载了，或者端口冲突等原因。其余回调都与 TcpConnect 相关。

5.1、EventLoop loop_

关联一个事件循环，在构造时传入，在创建 Accept 和创建 TcpConnect 时作为参数传入。

5.2、Accept accept_

负责监听一个 Ip 地址和端口号,在调用 void start()接口时开始监听。调用 void stop()接口停止监听。

5.3、TcpConnectMap connec_map_

TcpConnect 对象的集合采用 std::map 管理，key 是 TcpConnect 绑定的套接字文件描述符 fd。

每当 Accept 对象监听到连接请求时会调用 void newConnection(FD fd, SocketAddrIn& peer_addr)接口，接口中会创建一个 TcpConnect 对象，如果连接成功则添加到 map 中。

void newConnection(FD fd, SocketAddrIn& peer_addr)

Accept对象监听到连接请求时调用，函数中会创建TcpConnect对象并设置TcpMessageCallback message_callback_，TcpCloseCallback close_callback_，TcpErrorCallback error_callback_三个回调函数。而后通过TcpConnect::connectEstablished()接口尝试监听可读事件，如果监听成功则添加到connec_map_中管理并调用connection_callback_回调函数，否则说明服务器满载，调用full_load_callback_回调函数，并释放TcpConnect对象关闭该连接。

void closeConnection(TcpConnect* connect)

当对端端口连接或则服务器主动断开连接时调用，会调用回调函数 TcpCloseCallback close_callback_，而后将 connect 对象从 connec_map_取出并释放。

5.4、Buffer broadcast_buf_

用于 tcp 广播的 buffer，通过 void broadcast(const char *data, int len)接口将要广播的数据添加到 buffer 中，broadcast(……)函数会把 buffer 中的数据发送给每一个 connect_map_中的 tcp 连接。

5.5、Function Callback

NewTcpConnectCallback connection_callback_

Accept 监听到连接请求，创建新的 TcpConnect 并监听成功后被调用。

TcpMessageCallback message_callback_

在 void newConnection(FD fd, SocketAddrIn& peer_addr)接口中被调用，该回调会被注册到每个连接到当前服务器的 TcpConnect 的 TcpMessageCallback 中，任意一个 TcpConnect 有可读事件时都会被调用。

TcpCloseCallback close_callback_

在 void closeConnection(TcpConnect* connect)接口中被调用，任意一个 TcpConnect 将要关闭前被调用。

TcpErrorCallback error_callback_

在 void newConnection(FD fd, SocketAddrIn& peer_addr)接口中被调用，该回调会被注册到每个连接到当前服务器的 TcpConnect 的 TcpErrorCallback 中，任意一个 TcpConnect 有异常事件时都会被调用。

TcpServerFullLoadCallback full_load_callback_

在 void newConnection(FD fd, SocketAddrIn& peer_addr)接口中被调用，当有新的 TcpConnect 创建但是当前服务器已经满载而不能加入监听时被调用。

TcpServerStartErrorCallback start_error_callback_

在 void start()接口中被调用，当 TcpServer 开始监听端口失败时被调用，失败原因一般是因为服务器满载或者端口冲突。