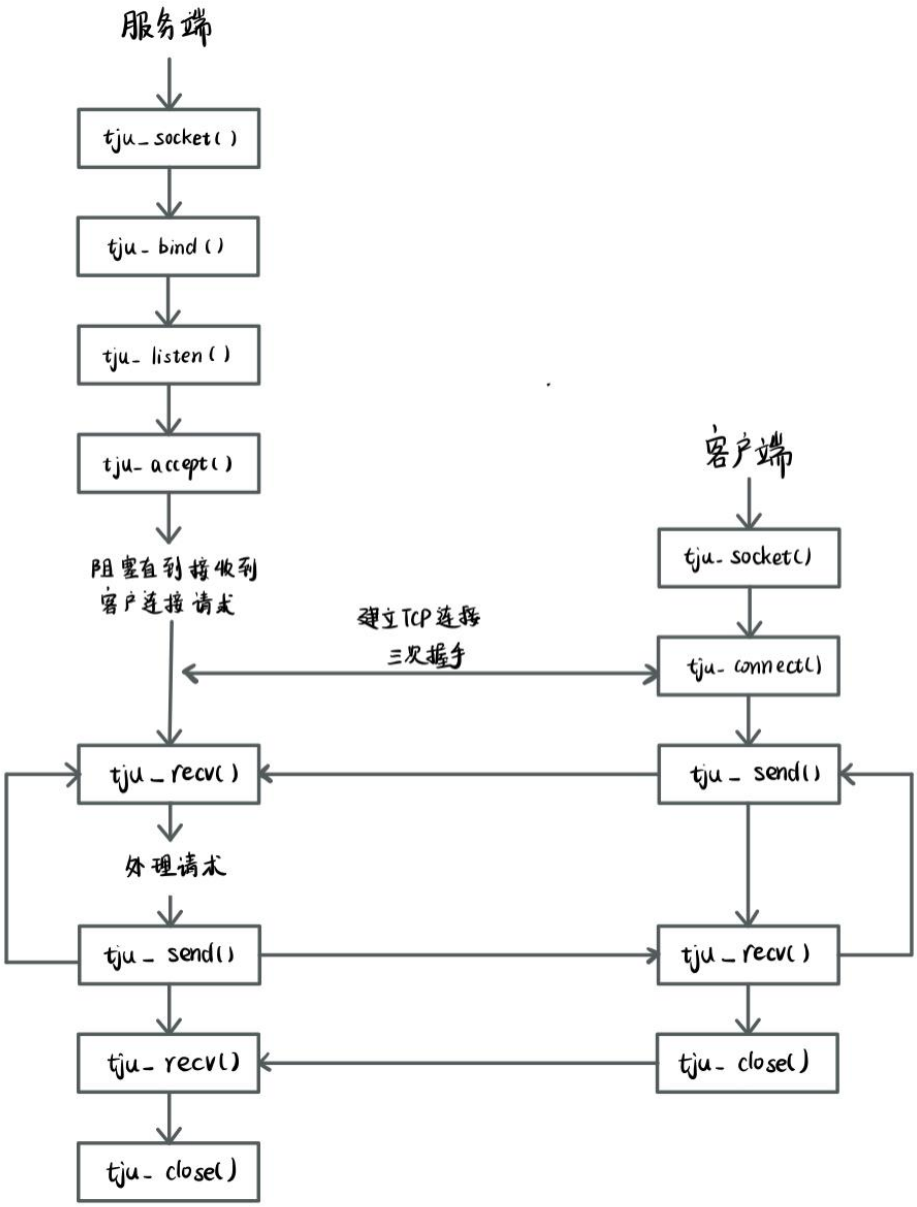
框架代码分析

首先，根据 socket 编程流程画出了如下流程图，该图包了含了本次实践项**目的核心函数**。 server.c **和** client.c **的主函数中各个** socket 编程接口的使用顺序**与下**图大致一致：



再根据如上的调用顺序对各个函数进行详细的分析，罗列出各个函数实现的内

部功能，被调用顺序等等。

为了建立 TCP 连接，我们必须创建相应的 TCP 套接字。所以，我们首先要做的就是在服务器端创建一个套接字，同时，在客户端创建另一个套接字。在这一步中，通信双方调用的函数是相同的，即都是 tju_socket()函数。

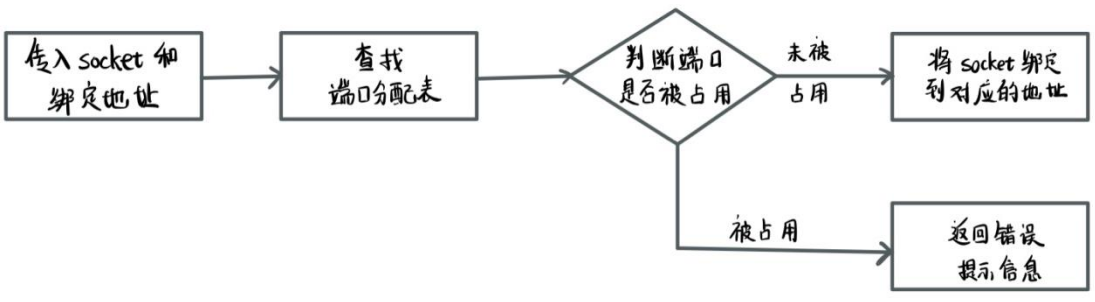
1.tju_socket( )

tju_socket()函数的作用就是生成一个用于通信的套接字。这个套接字可以作为稍后 tju_bind()函数的绑定对象。该函数实现的内容如下：

在绑定之前，我们用 tju_socket( )创建的套接字虽然存在于给定的空间中（底层数据结构已经就绪），但尚未为其分配相应的地址。为了能够正常使用 socket，接下来服务端需要做的是为建立好的 socket 分配相应的地址。

2.tju_bind( )

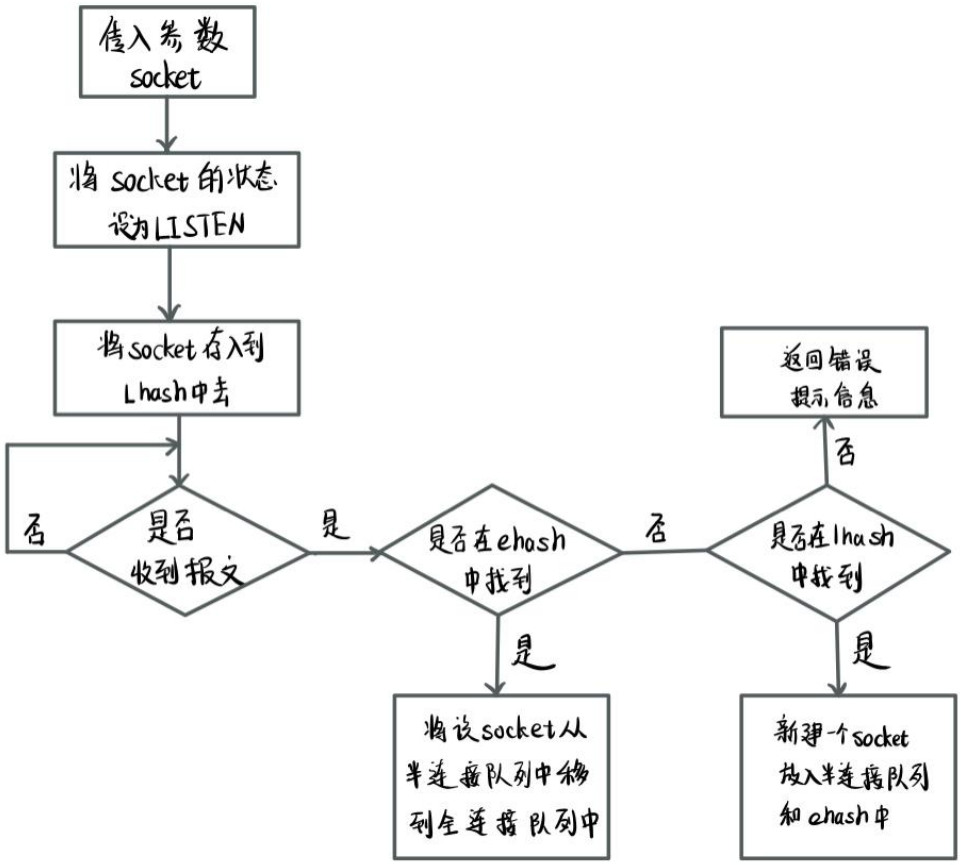
tju_bind( ) 函数的作用是将套接字绑定一个 IP 地址和端口号。在把 socket 绑定到相应的地址前需要查找端口分配表 bhash，以确保同一个地址不会与多个 socket 绑定。该函数实现的内容如下：



一旦我们将套接字与地址绑定起来后，接下来就是让它扮演服务器或客户端的角色。也就是说，它需要将自己设置为侦听传入的连接，或者启动与正在侦听的其他人的连接。将其设置为监听状态需要通过 tju_listen（ ）函数来实现。

3.tju_listen( )

绑定了端口的套接字可以作为 tju_listen()函数的监听对象。调用 tju_listen()之后，套接字就从 CLOSE 状态转变为 LISTEN 状态，于是这个套接字就可以对外提供 TCP 连接的窗口了。该函数实现的内容如下



* 补充：内核为任何一个给定的监听套接字维护两个队列：

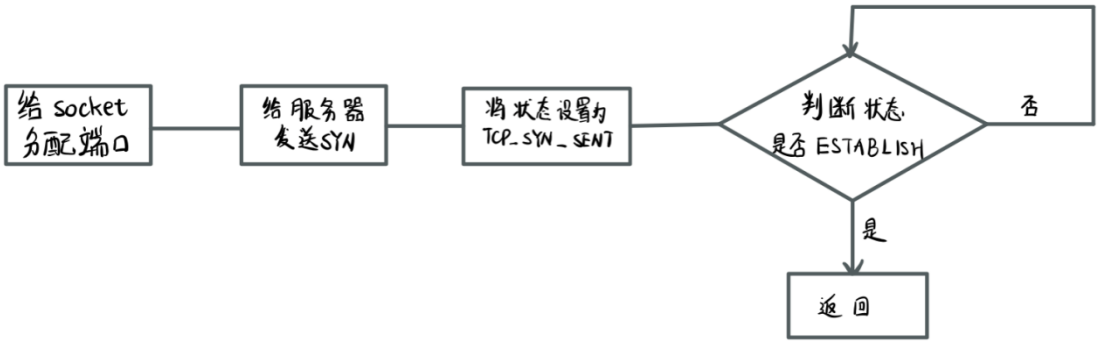
①未完成连接队列，每个这样的 SYN 分节对应其中一项：已由某个客户发出并到达服务器，而服务器正在等待完成相应的 TCP 三路握手过程。这些套接字处于 SYN_RECV 状态

② 已完成连接队列，每个已完成 TCP 三路握手过程的客户对应其中一项。这些套接字处于 ESTABLISHED 状态。

当服务端调用 tju_listen()函数进入监听状态以后，就会开始监听是否有客户端想要与其建立连接。如果客户端想要建立连接就需要调用 tju_connect()函数来实现。

4.tju_connect()

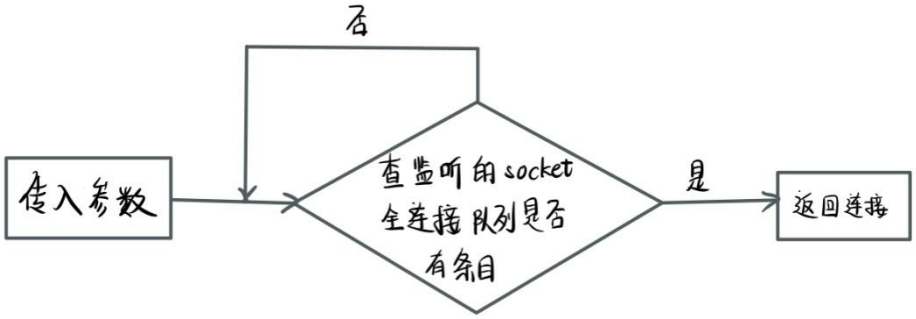
客户端在调用 tju_socket()函数来创建 socket 后，如果想要与服务端建立连接，就需要调用 tju_connect()函数来实现。该函数实现的内容如下：



在调用完 tju_listen（ ）后别调用的函数就是 tju_accept()。关于该函数的内容如下。

5.tju_accept()

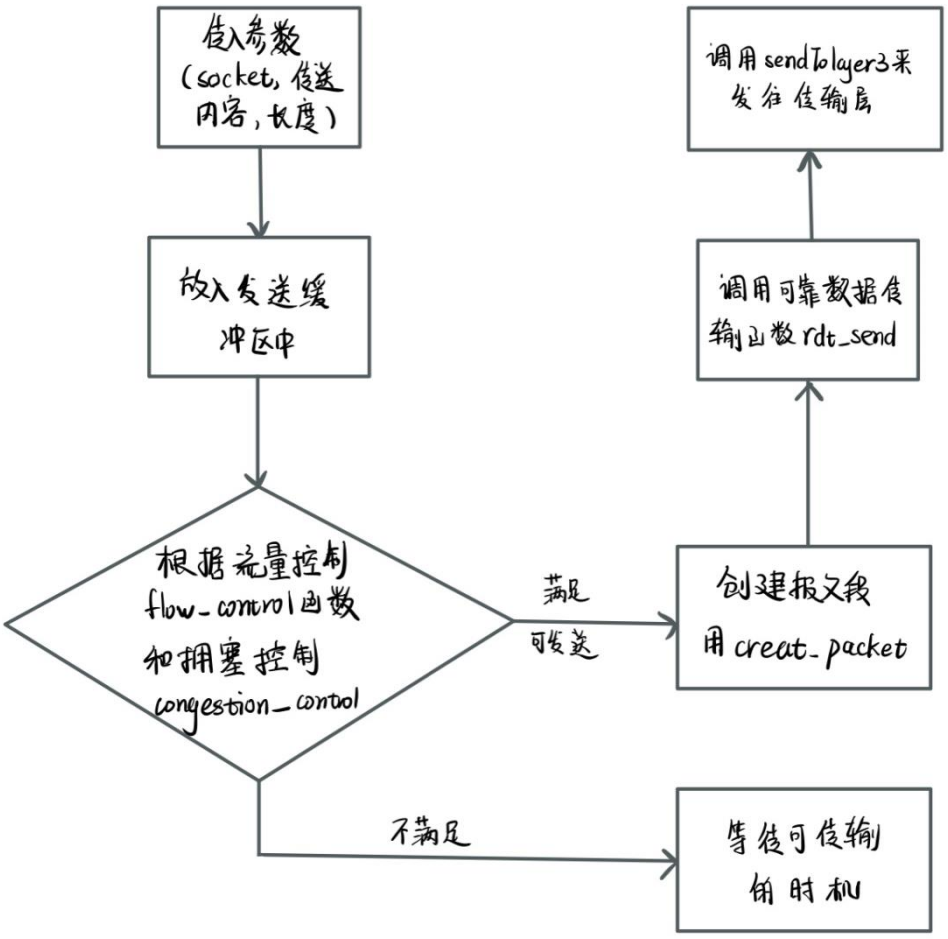
服务端调用 tju_accept 函数从处于监听状态的流套接字的全连接队列中取出排在最前的一个客户请求，返回新创建的套接字的描述符。该函数实现的过程如下：



经过以上几个函数的调用和执行后，**我**们就完成了"**三次握手**"，**建立了** TCP 连接。**往后就可以用**该 TCP 连接来进行消息的发送和接收。

6.tju_recv ()和 tju_sent()

tju_recv ()**和** tju_sent()**函数的**实现涉及到**可靠数据**传输、**流量控制**、拥塞控**制等多种机制**，**因此在**实现上比起其它函数来说更为复杂。**所以在此**阶段，**我只是整理出了**总体框架，**里面涉及到函数的具体具体内容我会随着**实践项目的**展开逐**渐完善。**并在**实践报告中各处设计流程，现阶段的流程图如下：



最后客户端调用 tju_close（ ）**函数来关**闭现有的连接，该过程涉及到了 TCP 连接管理中的"**四次**挥手"。

从 TCP 连接的建立到结束，客户端和服务端一共会经历 10 中状态，状态之间的转换对应了各个函数调用和交互，如下图所示：