

## 09 | 答疑篇：学习网络编程前，需要准备哪些东西？

2019-08-21 盛延敏

网络编程实战

[进入课程 >](#)



讲述：冯永吉

时长 10:43 大小 9.82M



你好，我是盛延敏，这里是网络编程实战第 9 讲，欢迎回来。

今天是基础篇的最后一讲。在这一讲中，我将会针对基础篇中大家提出的普遍问题进行总结和答疑，让我们整理一下，再接着学习下一个模块的内容。

### 代码和环境

既然我希望通过学习，可以带你进行网络编程实战，那么就得有一个环境，可以运行文章中的例子，并加以活学活用。

我已经将代码上传到 GitHub 中，你可以访问以下地址来获得最新的代码。


代码按照章节组织，比如 chap-7 就对应第七篇文章。

代码按照 CMake 组织，CMake 是一个跨平台的编译管理系统，使用 CMake 可以方便地在 Linux 等类 UNIX 系统下动态生成 Makefile，再由 make 工具编译、链接生成二进制文件。当然，CMake 也可以支持 Windows 系统下的 C/C++ 编译，这里我们就不展开了。

所有的代码我都已经测试过，可以运行在 Linux 和 MacOS 上。


## Ubuntu 系统

在 Linux 下，如果你是 Ubuntu 系统，需要安装 Cmake、make 和 gcc/g++ 等编译系统和工具。

 复制代码


```
1 sudo apt-get install gcc g++ make cmake
```

如果是 CentOS 或 Red Hat，需要执行 yum install 命令：

 复制代码


```
1 sudo yum install gcc g++ make cmake
```

使用 CMake 编译程序需要两步，第一步执行 Cmake 生成配置文件，主要是 Makefile；具体做法是执行如下的 cmake 命令，之后在 build 目录下，会发现 CMake 根据系统环境如编译器、头文件等自动生成了一份 Makefile：

 复制代码

```
1 cd build && cmake -f ../
```

接下来执行第二步，在 build 目录运行 make，让 make 驱动 gcc 编译、链接生成二进制可执行程序，这个过程可能会持续几分钟。最后在 build/bin 目录下，会生成所有可运行的二进制程序。


 复制代码

```
1 -rwxr-xr-x 1 vagrant vagrant 13944 Aug 18 13:45 addressused*
2 -rwxr-xr-x 1 vagrant vagrant 14000 Aug 18 13:45 addressused02*
3 -rwxr-xr-x 1 vagrant vagrant 13848 Aug 18 13:45 batchwrite*
4 -rwxr-xr-x 1 vagrant vagrant 13800 Aug 18 13:45 bufferclient*
5 -rwxr-xr-x 1 vagrant vagrant 14192 Aug 18 13:45 graceclient*
6 -rwxr-xr-x 1 vagrant vagrant 14096 Aug 18 13:45 graceserver*
7 -rwxr-xr-x 1 vagrant vagrant 8960 Aug 18 13:45 make_socket*
8 -rwxr-xr-x 1 vagrant vagrant 13920 Aug 18 13:45 pingclient*
9 -rwxr-xr-x 1 vagrant vagrant 14176 Aug 18 13:45 pingserver*
10 -rwxr-xr-x 1 vagrant vagrant 13976 Aug 18 13:45 reliable_client01*
11 -rwxr-xr-x 1 vagrant vagrant 13832 Aug 18 13:45 reliable_client02*
12 -rwxr-xr-x 1 vagrant vagrant 14120 Aug 18 13:45 reliable_server01*
13 -rwxr-xr-x 1 vagrant vagrant 14040 Aug 18 13:45 reliable_server02*
14 -rwxr-xr-x 1 vagrant vagrant 14136 Aug 18 13:45 samplebuffer01*
15 -rwxr-xr-x 1 vagrant vagrant 13864 Aug 18 13:45 samplebuffer02*
16 -rwxr-xr-x 1 vagrant vagrant 14392 Aug 18 13:45 samplebuffer03*
17 -rwxr-xr-x 1 vagrant vagrant 13848 Aug 18 13:45 streamclient*
18 -rwxr-xr-x 1 vagrant vagrant 14392 Aug 18 13:45 streamserver*
19 -rwxr-xr-x 1 vagrant vagrant 13784 Aug 18 13:45 tcpclient*
20 -rwxr-xr-x 1 vagrant vagrant 13856 Aug 18 13:45 tcpserver*
21 -rwxr-xr-x 1 vagrant vagrant 13936 Aug 18 13:45 udpclient*
22 -rwxr-xr-x 1 vagrant vagrant 13320 Aug 18 13:45 udpserver*
23 -rwxr-xr-x 1 vagrant vagrant 13936 Aug 18 13:45 unixdataclient*
24 -rwxr-xr-x 1 vagrant vagrant 13896 Aug 18 13:45 unixdataserver*
25 -rwxr-xr-x 1 vagrant vagrant 13800 Aug 18 13:45 unixstreamclient*
26 -rwxr-xr-x 1 vagrant vagrant 13992 Aug 18 13:45 unixstreamserver*
```

## MacOS

在 MacOS 上，Cmake 和 make 都会有 Mac 特定版本，并且实现的原理也是基本一致的，我们可以像上面 Ubuntu 系统一样手动安装、配置这些工具。


如果你的系统上没有这两个软件，可以使用 brew 安装 Cmake 和 make。

 复制代码

```
1 brew install cmake
2 brew install make
```


MacOS 上 C/C++ 语言的编译器不同于 GNU-GCC，是一个叫做 Clang 的东西。Clang 背后的技术叫做 LLVM ( Low Level Virtual Machine )。LLVM 是以 BSD License 开发的开源编译器框架系统，基于 C++ 编写而成，不仅可以支持 C/C++，还可以支持 Swift、Rust 等语言。

如果你在 MacOS 上查看 Clang 的版本信息，可以很明显地看到，Clang 是基于 LLVM 开发的，并且对应的版本是多少。在我的机器上显示的 LLVM 版本是 10.0.0。

 复制代码

```
1 clang -v
2 Apple LLVM version 10.0.0 (clang-1000.10.44.4)
3 Target: x86_64-apple-darwin17.7.0
4 Thread model: posix
5 InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

下面是在 MacOS 上执行 Cmake 和 make，使用 Clang 完成编译和链接的过程。

 复制代码

```
1 cd build && cmake -f ../
2 -- The C compiler identification is AppleClang 10.0.0.10001044
3 -- The CXX compiler identification is AppleClang 10.0.0.10001044
4 -- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc
5 -- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc -- worki
6 -- Detecting C compiler ABI info
7 -- Detecting C compiler ABI info - done
8 -- Detecting C compile features
9 -- Detecting C compile features - done
10 -- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c++
11 -- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c++ -- w
12 -- Detecting CXX compiler ABI info
13 -- Detecting CXX compiler ABI info - done
14 -- Detecting CXX compile features
15 -- Detecting CXX compile features - done
16 -- Configuring done
17 -- Generating done
18 -- Build files have been written to: /Users/shengym/Code/network/yolanda/test
```

```
1 cd build && make
```

[复制代码](#)

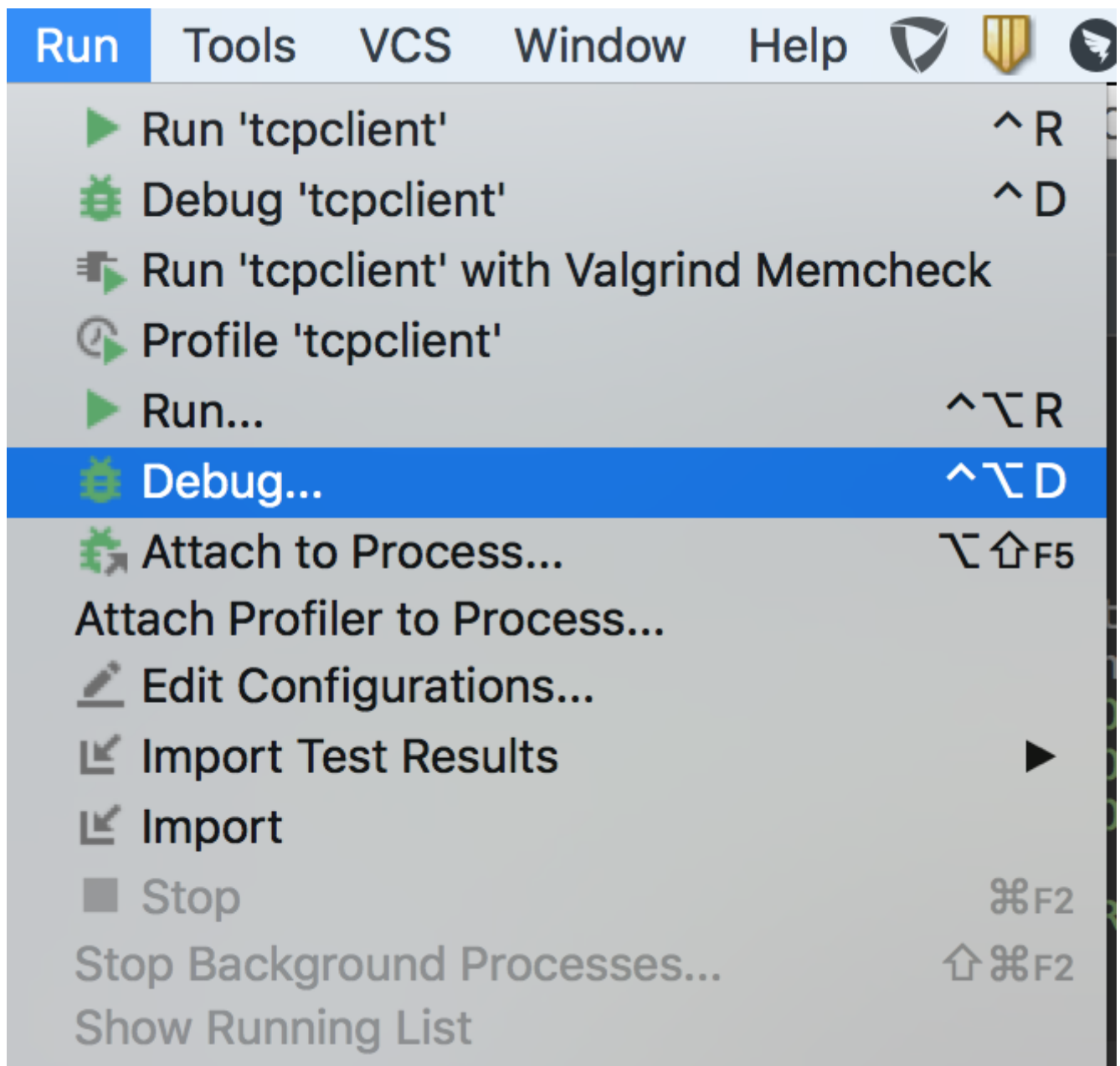
可执行程序仍然保存在 build/bin 目录下面。

## CLion

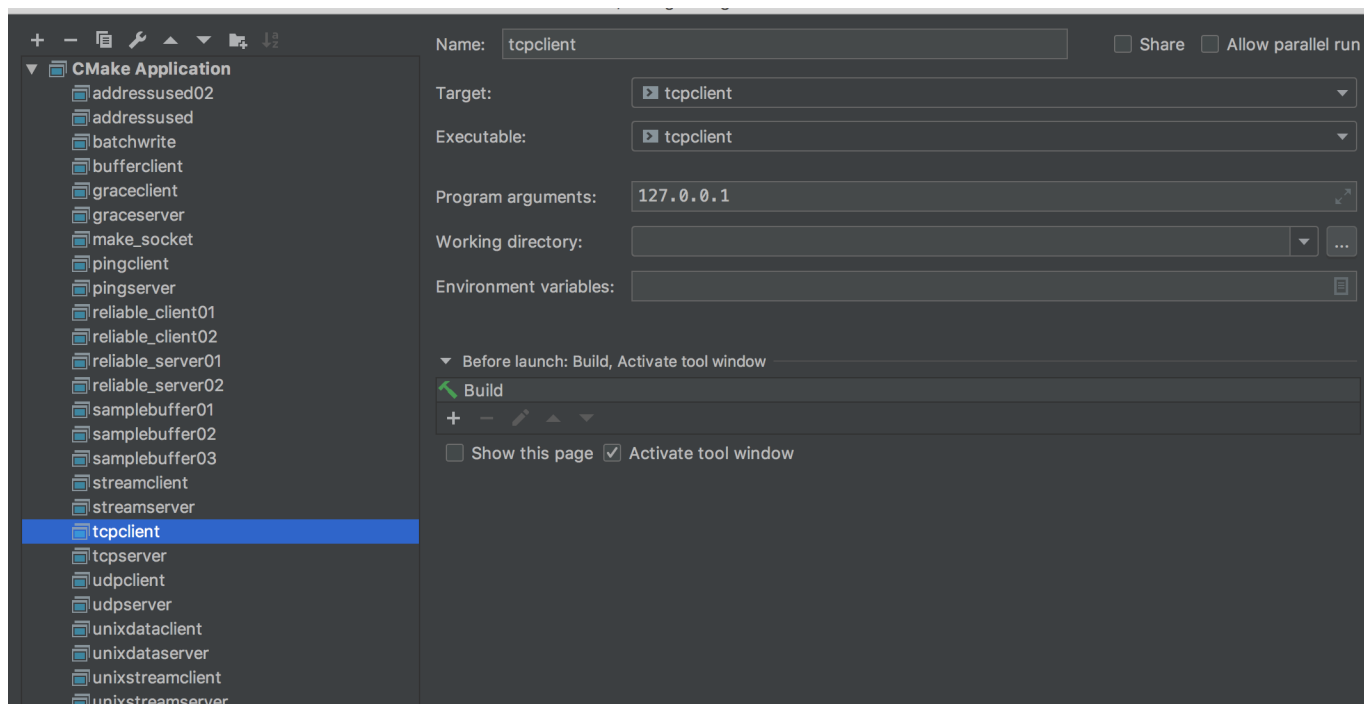
对于有 IDE 情结的同学来说，推荐使用 JetBrains 公司出品的 CLion 进行编译和调试。

你可以在这里下载 <https://www.jetbrains.com/clion/>，获得 30 天的免费使用。

CLion 自带了 CMake 等工具，可以开箱即用。它最强大的地方，是可以直接设置断点，方便进行调试。我们进入主菜单，选择 Run，再选择 Debug，就可以启动程序，进行调试。



有些情况下，启动程序时需要输入一些参数，这个时候需要使用“Edit Configurations”为可执行程序配置参数。下面是一个例子：



## 学习路径

也许你刚刚入门，一直对网络编程的学习路径有很大的困惑，我在这里统一回复一下。

我觉得学习网络编程技术，必须要过一道语言关，这个语言就是 C 语言。专栏本身也是基于 C 语言的。虽然现代工业化语言如 Java、Golang 等已经足够强大，但是你要知道，在这些语言的背后，无一例外的总是有 C 语言的影子。C 语言是可以和系统直接交互的语言，无论是系统调用，还是内核实现，都和 C 语言有非常直接的联系，比如 Java 本身就是用 C++ 实现的，Golang 虽然现在可以自举，也就是可以使用 Golang 实现 Golang 本身，但是它的第一版也是用 C/C++ 实现的。

我不建议一开始就学习 C++ 语言，在我看来，C++ 语言在 C 语言原来的基础上做了很多语言层面的增强，而这些增强的语言特性，例如模板、继承、虚函数、boost 语言库等，对于刚开始接触底层的人显得有些艰深。

学习一门编程语言，显然不是学习这门语言的控制流或者变量类型，而是抓住这门语言的精髓。我认为 C 语言的精髓包括数组和指针、结构体和函数。

C 语言的地址、数组、指针可以帮助我们详细地理解计算机的体系结构，一段数据怎样在内存中摆放，怎么去访问等，你可以在学习它们的过程中得到锤炼，了解这些基础的编程理念。



有些同学一上来就啃“TCP/IP 协议”，我觉得对于实战来说，显得过于着急。我们可以把“TCP/IP 协议”当做编程过程中答疑解惑的好帮手，有问题之后再从中寻找答案，而不是急急忙忙就来啃这类书籍。说实话，这类书籍理论性偏强，有时候大段读下来也少有收获。

最好的办法，还是自己跟随一些入门书籍，或者我的这篇实战，尝试动手去写、去调试代码，这中间你会不断获得一些反馈，然后再和大家一起探讨，不断加深了解。

当你学到了一定阶段，就可以给自己开一些小的任务，比如写一个聊天室程序，或者写一个 HTTP 服务器端程序，带着任务去学习，获得成就感的同时，对网络编程的理解也随之更上一层楼了。

## 书籍推荐

我希望你可以通过这个专栏更好地了解网络编程，但是深入的学习还需要你自行去找更多的资料。我在这里给你推荐一些书，这些书是各个领域的经典。

C 语言入门方面，我推荐《C 程序设计语言》，我在文稿里附上了豆瓣链接，你可以看下大家的评价以及他们的学习方式：<https://book.douban.com/subject/1139336/>

UNIX 网络编程方面，强烈推荐 Stevens 大神的两卷本《UNIX 网络编程》，其中第一卷是讲套接字的，第二卷是讲 IPC 进程间通信的。这套书也随书配备了源代码，你如果有兴趣的话，可以对代码进行改写和调试。

豆瓣链接在此：<https://book.douban.com/subject/1500149/>

这套书的卷一基本上面面俱到地讲述了 UNIX 网络编程的方方面面，但有时候稍显啰嗦，特别是高性能高并发这块，已经跟不上时代，但你可以把注意力放在卷一的前半部分。

这套书翻译了好几版，就我的体验来说，比较推荐杨继张翻译的版本。

TCP/IP 协议方面，当然是推荐 Stevens 的大作《TCP/IP 详解》，这套书总共有三卷，第一卷讲协议，第二卷讲实现，第三卷讲 TCP 事务。我在这里推荐第一卷，第二卷的实现是基于 BSD 的代码讲解的，就不推荐了。我想如果你想看源码的话，还是推荐看 Linux 的，毕竟我们用的比较多。第三卷涉及的内容比较少见，也不推荐了。



这套书各个出版社翻译了好多版本，你可以去豆瓣自行查看哪个版本评分比较高。

《TCP/IP 详解 卷 1：协议》豆瓣链接如下：

<https://book.douban.com/subject/1088054/>

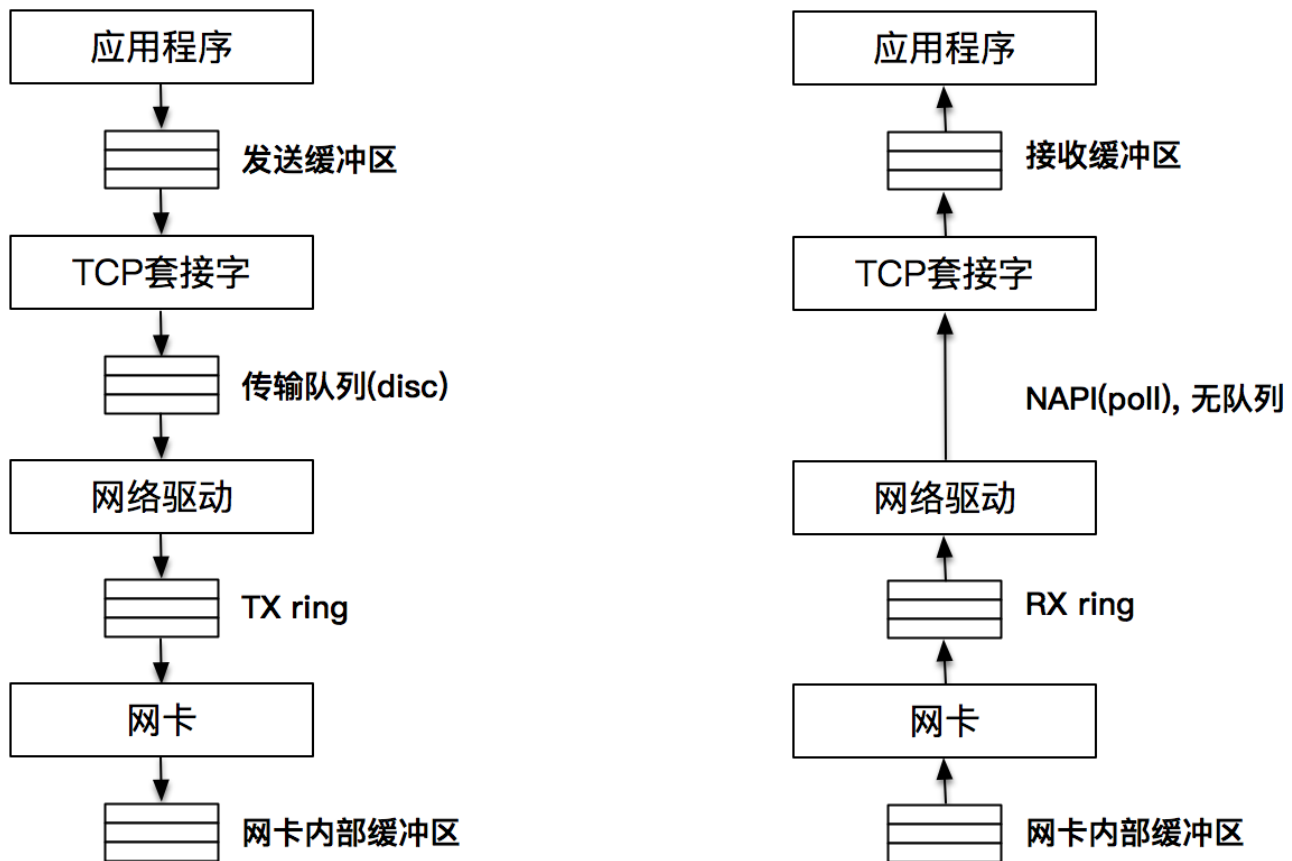
最后除了书籍外，还有一个非常好的了解 TCP 的方法，那就是查看 RFC 文档，对于有一定英文能力的同学来说，可以说是一个捷径。RFC 最大的好处可以帮我们了解 TCP 发展的背景和脉络。

## 疑难解答

前面的内容算是我对你学习网络编程提供的一些小建议或者小帮助。接下来，我们正式进入到文章本身的内容。

在第 5 讲思考题部分中，我出了这么一道题目“一段数据流从应用程序发送端，一直到应用程序接收端，总共经过了多少次拷贝？”大家的回答五花八门。

我的本意可以用文稿中的一张图来表示，还记得 TCP/IP 层次模型么？我想通过这么一个问题，来展示 TCP/IP 分层的思想。



让我们先看发送端，当应用程序将数据送到发送缓冲区时，调用的是 send 或 write 方法，如果缓存中没有空间，系统调用就会失败或者阻塞。我们说，这个动作事实上是一次“显式拷贝”。而在这之后，数据将会按照 TCP/IP 的分层再次进行拷贝，这层的拷贝对我们来说就不是显式的了。

接下来轮到 TCP 协议栈工作，创建 Packet 报文，并把报文发送到传输队列中（qdisc），传输队列是一个典型的 FIFO 队列，队列的最大值可以通过 ifconfig 命令输出的 txqueuelen 来查看。通常情况下，这个值有几千报文大小。

TX ring 在网络驱动和网卡之间，也是一个传输请求的队列。

网卡作为物理设备工作在物理层，主要工作是把要发送的报文保存到内部的缓存中，并发送出去。

接下来再看接收端，报文首先到达网卡，由网卡保存在自己的接收缓存中，接下来报文被发送至网络驱动和网卡之间的 RX ring，网络驱动从 RX ring 获取报文，然后把报文发送到上层。

这里值得注意的是，网络驱动和上层之间没有缓存，因为网络驱动使用 Napi 进行数据传输。因此，可以认为上层直接从 RX ring 中读取报文。

最后，报文的数据保存在套接字接收缓存中，应用程序从套接字接收缓存中读取数据。

这就是数据流从应用程序发送端，一直到应用程序接收端的整个历程，你看懂了吗？

上面的任何一个环节稍有积压，都会对程序性能产生影响。但好消息是，内核和网络设备供应商已经帮我们把一切都打点好了，我们看到和用到的，其实只是冰山上的一角而已。

这就是基础篇的总结与答疑部分，我先对之前基础篇的内容补充了一些资料，尽可能地为你学习网络编程提供方便，然后针对大家有明显疑惑的问题进行了解答，希望你有所帮助。



# 网络编程实战

从底层到实战，深度解析网络编程

盛延敏

前大众点评云平台首席架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 08 | 工欲善其事必先利其器：学会使用各种工具

下一篇 10 | TIME\_WAIT：隐藏在细节下的魔鬼

## 精选留言 (15)

写留言



ly

2019-08-21

哈哈，和老师灵魂碰撞了，unix的网络编程卷1和2昨天下单的。  
按照第5章的相关逻辑自己用Java写了一段demo程序。

【Java代码逻辑】

服务端接收 1024 字节(read)就休眠1秒钟；

客户端for循环50次,每次for循环写 10 \* 1024 字节出去(write/flush)。...

展开 ▾



4



石将从

2019-08-21

老师，你这个环境省略了好多东西啊，cd build && cmake -f ../  
哪里的Build目录呀

展开 ▾



1



传说中的成大大

2019-08-21

那个build目录是那个路径下的啊？我感觉都是系统路径下呢



1



无名

2019-08-25

所以总共经过了7次拷贝？

展开 ▾



沉淀的梦想

2019-08-23

网卡一般是处理到IP层吗？

展开 ▾



Geek\_Claire

2019-08-22

tx ring也算是一次拷贝吗？

展开 ▾



**Over10@d**

2019-08-22

根据最后一张图，对于接收方，是必须要等到数据存入到最上面的接收缓冲区后，操作系统才会发送ACK？还是当数据从Rx ring取出后ACK就可以发送了？或者是必须要等应用程序将数据从接收缓冲区读出后才发出？



**石将从**

2019-08-21

```
[root@izm5eb4ves923h7tl63px4z build]# cmake -f ../
CMake Error: Could not find CMAKE_ROOT !!!
CMake has most likely not been installed correctly.
Modules directory not found in
/app/cmake/share/cmake-3.15...
```

展开 ▾

1



**传说中的成大大**

2019-08-21

所以最终是拷贝了六次？

展开 ▾



**张立华**

2019-08-21

cmake的 -f参数，干什么的。老师，我google查了下，都没找到这个参数啊

1



**许童童**

2019-08-21

一段数据流从应用程序发送端，一直到应用程序接收端，总共经过了多少次拷贝？这道思考题确实很难，网上搜到的结果也是五花八门。

展开 ▾





马仔  
2019-08-21

我想学习Linux代码在window上的porting，但是网上的资料都比较散。老师可以推荐一些比较完整的资料吗。

展开 ▾



摘星星的人  
2019-08-21

运行起来了，好多警告，是 gcc 版本问题吗，gcc7.4，wsl Ubuntu18.04。根据源码，好好学习，还是有点懵懂的

展开 ▾



微笑  
2019-08-21

transferTo方法是节省了中间两次数据的copy吗？

展开 ▾



业余爱好者  
2019-08-21

搞java的，看了这篇，更有信心学好c和网络编程了

展开 ▾

