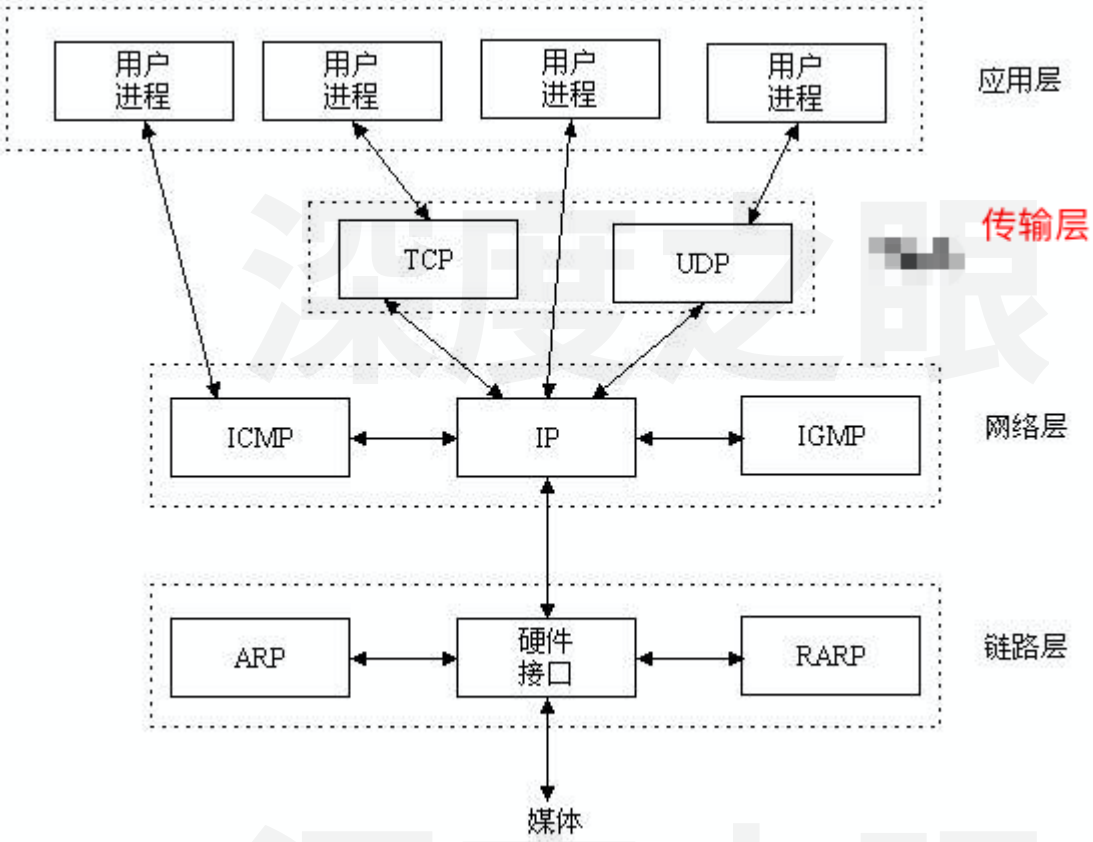
第十九章套接编程

> 本文是Python通用编程系列教程，已全部更新完成，实现的目标是从零基础开始到精通Python编程语言。本教程不是对Python的内容进行泛泛而谈，而是精细化，深入化的讲解，共5个阶段，25章内容。所以，需要有耐心的学习，才能真正有所收获。虽不涉及任何框架的使用，但是会对操作系统和网络通信进行全局的讲解，甚至会对一些开源模块和服务器进行重写。学完之后，你所收获的不仅仅是精通一门Python编程语言，而且具备快速学习其他编程语言的能力，无障碍阅读所有Python源码的能力和对计算机与网络的全面认识。对于零基础的小白来说，是入门计算机领域并精通一门编程语言的绝佳教材。对于有一定Python基础的童鞋，相信这套教程会让你的水平更上一层楼。

一 套接字层介绍

1. 套接字层的由来

通过前面章节的学习，你明白了计算机网络，计算机硬件和写软件怎么写，那么接下里我们就会使用所学过的东西来写一个客户端与服务端软件。按照网络协议把数据组织好，再沿着网络协议发送出去，送给对方之后在按照网络协议把数据解包，这样就会到达了对方的软件。



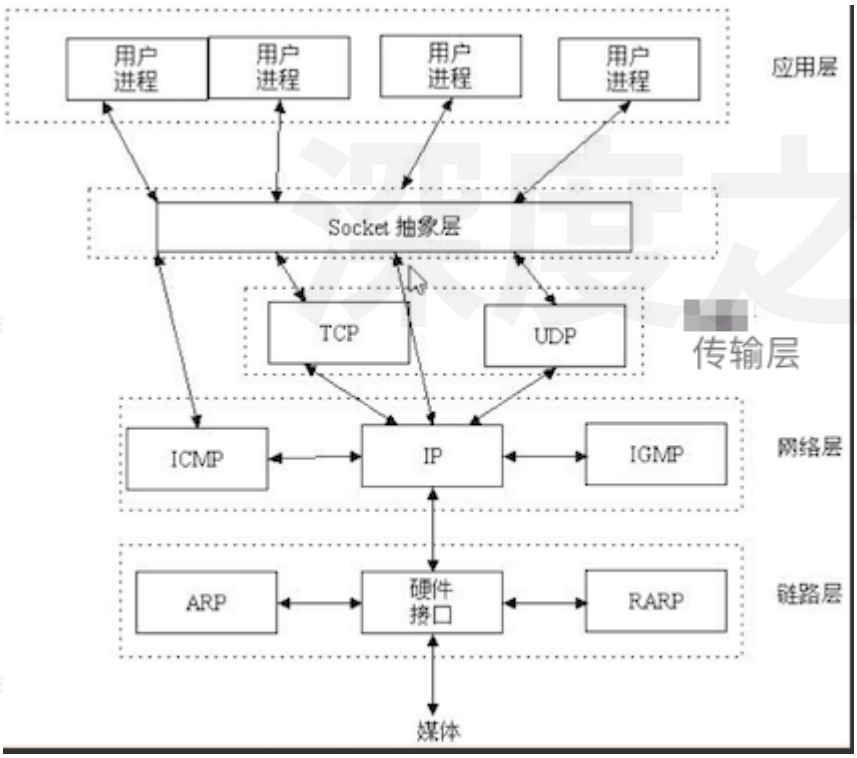
以后我们的工作在应用层，应用层产生数据交给下面的传输层去负责传输 ，你需要控制你的操作系统与目标主机建立三次握手和四次挥手，还需要控制操作系统把数据的格式封装好，先封装传输层的head，再封装网络层的head，再封装数据链路层的head，再转成二进制调网卡发送数据。如果按照这种方式写程序，你写一个TCP通信可能就要半年，软件核心逻辑还没写好，你就被开除了，紧接着你们公司就倒闭了。我们在文件操作那一章节操作硬盘的时候是用的open，并没有直接操作硬件，open本身就是硬盘高度封装的给我们使用的结果，这就像是一个接口，我们可以直接使用。同理， 我们可以使用别人写好的代码来完成这个通信的过程，在IT届有一个说法是避免重复造轮子，我们可以使用别人已经造好的轮子，但是要想称为真正的高手造轮子就是你的必经之路。这套课程，我们不教造轮子。

如果我们要基于网络通信写程序，要弄明白TCP/UDP协议，ip协议和以太网协议，那就真的要把每一层的协议都研究明白， 并不能说研究不明白，TCP/IP协议你研究一两年都不敢说自己精通，TCP协议自从网络开始就一直存在，这个协议非常古老，非常重要，重要到就像阳光和雨露一样，大家有时候甚至都可以把它忽略，在这样的前提下，创新的可能性非常低，你学习了当然有好处，但是没有必要详细的去了解。所以，对于软件开发来说，最好就是只负责处理应用层，传输层和传输层以下最好是封装好，只要了解工作原理就好了。

2. 套接字层

Socket层就是套接字层，套接字层在应用层和传输层之间，它的作用就是把传输层和传输层以下的协议都封装好， 所以以后写程序我们把应用层软件写好了之后，只要遵循套接字层的标准写出来的程序自然就是

遵循TCP/UDP和ip协议和以太网协议的标准的。那套接字协议的标准是什么标准呢？非常简单，只需要记住几个接口就可以了。



3. Socket层介绍

Socket是应用层与TCP/IP协议族通信的中间的软件抽象层，它是一组接口。在设计模式中，Socket其实就是一个门面模式，他把复杂的TCP/IP协议隐藏在接口后面，对于用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议。

> 利用套接字写出来的程序是希望能在客户端和服务端之间实现通信，
> 客户端和服务端的通信必不可少的就是ip和端口，
> 所以，一提到套接字，有人会说就是一堆ip和端口，
> 指的是套接字的工作原理，客户端和服务端都要有一个对应的ip和端口

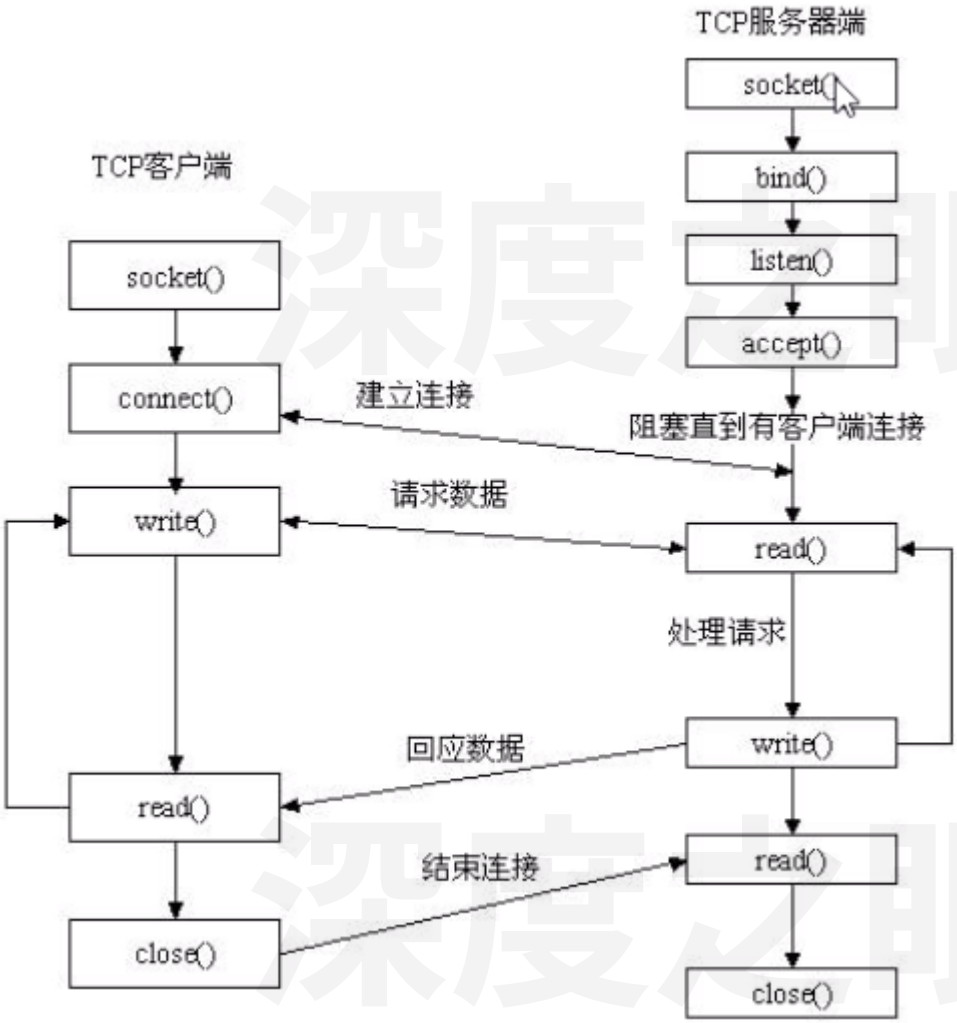
套接字起源于 20 世纪 70 年代加利福尼亚大学伯克利分校版本的 Unix,即人们所说的 BSD Unix。 因此,有时人们也把套接字称为“伯克利套接字”或“BSD 套接字”。一开始,套接字被设计用在同 一台主机上多个应用程序之间的通讯。这也被称进程间通讯,或 IPC。套接字有两种（或者称为有两个种族）,分别是基于文件型的和基于网络型的。

基于文件类型的套接字家族套接字家族的名字：AF_UNIXunix一切皆文件，基于文件的套接字调用的就是底层的文件系统来取数据，两个套接字进程运行在同一机器，可以通过访问同一个文件系统间接完成通信

> **基于网络类型的套接字家族**套接字家族的名字：AF_INET(还有AF_INET6被用于ipv6，还有一些其他的地址家族，不过，他们要么是只用于某个平台，要么就是已经被废弃，或者是很少被使用，或者是根本没有实现，所有地址家族中，AF_INET是使用最广泛的一个，Python支持很多种地址家族，但是由于我们只关心网络编程，所以大部分时候我们只使用AF_INET)

4. Socket工作流程

这是一个基于TCP通信的客户端和服务端，我们先来看服务端，以前学过面向对象，socket其实就是一个类，加括号调用这个类，就是实例化产生一个套接字对象。接下来是绑定，客户端和服务端都有ip和端口，对于服务端来说是提供服务的，那么就必须要有一个固定的地址，所以服务端的ip和端口需要与服务端机器绑定，这也就是宣告了我的服务端机器在这里，你们所有的客户端机器就来向我这个地址发送请求就可以了。下一步是监听，也就是服务端需要监听客户端发过来的syn请求，再往下面是accept，也就是接收客户端的连接。当客户端发送过来一个connect，正好对应服务端的accept。



基于TCP协议的数据通信需要通过三次握手建立连接，有了套接字之后非常简单，你只需要记住客户端的connect和服务端的accept，他们在底层就是客户端与服务端三次握手建立连接。建立好了双向连接之后，客户端与服务端就开始通信了，客户端可以发送数据给服务端，服务端也可以发送数据给客户端，这就是一个通信循环，完了之后客户端可以关连接，服务端也可以关连接。

二 套接字编程

1. Socket编程

基于套接字来写程序，以后必须要有一个服务端和一个客户端，接下来我们分别用代码来实现服务端和客户端。
一个生活中的场景，我们打电话的过程，要先有一个电话，然后是插卡，最后是拨号或者是待机等待别人拨号，电话接通后，这时你和你的朋友就建立起了连接，就可以讲话了。等交流结束，挂断电话结束此次交谈。
服务端代码

```
import socket
phone = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # 买电话
phone.bind(('127.0.0.1', 8082))  # 插手机卡，补充：0-65535 0-1024给系统用的
phone.listen(5)  # 进入待机状态，5指的是挂起连接数，就是backlog，指的是同一时间可以来五个请求
print('start...')
conn, client_address = phone.accept()  # 等电话连接
print('连接来了：', conn, client_address)
# 收发消息
msg = conn.recv(1024)  # 收消息，1024个字节（bytes）是一个最大的限制
print('客户端的消息：', msg)
conn.send(msg + b'Albert')
# 挂电话，回收系统资源
conn.close()
# 关机
phone.close()
```

conn是一个套接字对象，这就代表TCP三次握手的产物，是一个双向连接，client_address指的是客户端的ip和端口，对于服务端来说需要有一个固定的ip和端口，所以需要先绑定一下，然后等待客户端连接。
客户端代码

```
import socket
phone = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # 买电话
phone.connect(('127.0.0.1', 8082))  # 拨电话，地址为服务端的ip和端口
phone.send('你好'.encode('utf-8'))  # 发消息b'你好'
data = phone.recv(1024)  # 收消息
```

```
print(data.decode('utf-8'))
phone.close()
```

对于客户端来说，ip和端口不是需要固定的，所以并不需要绑定，只需要知道目标主机的ip和端口，直接拨号连接就好了
需要注意的是当服务端关掉再重启之后，我们写的程序会给操作系统发一个指令：清理刚才建立连接的资源，与机器的硬件配置相关，可能有的机器不会立即清理掉，就会提示端口被占用， 这时候换一个端口就可以了。

2. 通信循环

试想一个场景，半夜十二点，我半夜十二点给一个朋友打电话，我说了一个“你好”，他回了一个“你好，Albert”，然后两个人把电话就挂掉了，如果是这样，这两人肯定有毛病。
计算机通信的过程和打电话类似，肯定不能是发一个消息就结束了，应该可以不断多次的发消息和收消息，这就是通信循环。
服务端代码

```
import socket
phone = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
phone.bind(('127.0.0.1', 8082))
phone.listen(5)
print('start...')
conn, client_address = phone.accept()
print('连接来了：', conn, client_address)
while True:
    msg = conn.recv(1024)
    print('客户端的消息：', msg)
    conn.send(msg + b'SB')
conn.close()
phone.close()
```

客户端代码

```
import socket
phone = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
phone.connect(('127.0.0.1', 8082))
while True:
    msg = input('请输入你的名字>>:').strip()
    if msg in ['albert', 'Albert', 'mayite', '马一特']:
```

```
            msg = '机器禁止发送，请重新输入'
            print(msg)
            continue
        phone.send(msg.encode('utf-8'))
        data = phone.recv(1024)
        print(data.decode('utf-8'))
    phone.close()
```

现在如果我们客户端单方面关闭程序，对于服务端来说还会等待接收，但是客户端已经跑路了，服务端程序也就崩溃了，如果是这样的话，在淘宝天猫双十一的时候，你把程序突然关掉了，天猫淘宝的程序就瘫痪了，这是我们绝对不允许出现的，那么我们要怎么解决呢?
用户关掉客户端这就是一个异常，当出现可以预测的异常的时候，我们可以使用if来进行判断，但是当用户行为无法预测而这个异常又一定会发生的时候，我们使用try来进行异常的捕获。
服务端代码

```
import socket
phone = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
phone.bind(('127.0.0.1', 8083))
phone.listen(5)
print('start...')
conn, client_address = phone.accept()
print('连接来了：', conn, client_address)
while True:
    try:
        msg = conn.recv(1024)
        print('客户端的消息: ', msg)
        conn.send(msg + b'SB')
    except Exception:
        break
conn.close()
phone.close()
```

3. 连接循环

基于我们刚才写的通信循环，客户端退出了，服务器虽然没有报错，但是也跟着一块终结，这肯定是不合理的，我们需要的服务器是能够持续不断的为客户端机器提供服务的，某一个客户端退出了，这并不能影响服务端为其他的客户端提供服务，所以，我们还需要再加入一个连接循环。
服务端代码

```
import socket
phone = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
phone.bind(('127.0.0.1', 8081))
phone.listen(5)
print('start...')
while True:  # 连接循环
    conn, client_address = phone.accept()
    print('客户端 ', client_address)
    while True:  # 通信循环
        try:
            msg = conn.recv(1024)
            print('客户端的消息: ', msg)
            conn.send(msg + b'SB')
        except Exception:
            break
    conn.close()
phone.close()
```

客户端代码

```
import socket
phone = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
phone.connect(('127.0.0.1', 8081))
while True:
    msg = input('>>>: ').strip()
    if msg in ['albert', 'Albert', 'mayite', '马一特']:
        msg = '机器禁止发送，请重新输入'
        print(msg)
        continue
    phone.send(msg.encode('utf-8'))
    data = phone.recv(1024)
    print(data.decode('utf-8'))
phone.close()
```

服务端代码只需要一份，客户端代码你可以复制7份，先运行服务端，再按照客户端文件顺序依次运行七个客户端代码，你会发现你的服务端机器只能同时服务于一台客户端机器，只有第一台客户端机器关掉之后才能服务于第二胎客户端机器，这就是通信循环。除此之外，你还会发现第七个运行客户端是没有效果的，因为我们服务端最大挂起连接数是5，除了有一个正在服务的，已经挂起了5个连接请求了。

> 上面我们写的程序只能同时服务与一个用户，就像是一个小饭店只有一个服务员，来一个客人就先服务这个客人，后面还有5位客人在排队等着，屋里空间小，外面在下雨，第六个人不会在屋外面排队，只有当这个服务员把第一位客人服务走了之后，后面排队的客人才能坐下点餐吃饭。

你肯定会有疑问，那如何才能使服务端机器同时服务于多个客户端呢？这将是我们下一阶段并发编程要学习的内容。