Socket API

理解 Socket 首先要熟悉 TCP/IP 协议簇，TCP/IP(Transmission Control Protocol/Internet Protocol，传输控制协议/网间协议)定义了主机如何接入 Internet 及数据如何传输的标准。

TCP/IP 网络协议

TCP/IP 是 TCP 和 IP 协议的合称，但实际上 TCP/IP 协议是指 Internet 整个 TCP/IP 协议簇，不同于 ISO 七层模型的分层，TCP/IP 协议参考模型把所有 TCP/IP 系列协议归纳抽象为 4 层：

- 应用层 协议：TFTP、HTTP、SNMP、FTP、SMTP、DNS、Telnet...
- 传输层 协议：TCP、UDP
- 网路层 协议：IP、ICMP、OSPF、EIGRP、IGMP
- 数据链路层 协议：SLIP、CSLIP、PPP、MTU

TCP/IP 是一种工业标准的协议集，为广域网 WAN 而设计。每一个抽象层建立在低一层所提供的服务上，并为高一层提供服务。

在 TCP/IP 协议中两个 Internet 主机通过路由器和对应的层连接，各主机上得应用通过数据通道相互执行读取操作。

TCP 与 IP

本地进程间通信

”进程通信“的概念最初来源于单机系统，由于每个进程都在自己地址范围内运行，为保证两个相互通信的进程之间互不干扰又协调一致的工作，操作系统为进程通信提供了相应的设施。

本地进程间通信 IPC 方式

- 消息传递：管道、FIFO、消息队列
- 同步：互斥量、条件变量、读写锁、文件和写记录锁、信号量
- 共享内存：具名、匿名
- 远程过程调用：RPC

网络进程间通信

本地进程可通过进程号（Process ID, PID）唯一标识，网络中进程间如何通信呢？首先要解决的问题是**如何唯一标识一个进程**，其实网络中 TCP/IP 已经解决了这个问题。

- 网络层 IP 地址可以唯一标识网络中的主机
- 传输层 “协议+端口”可以唯一标识主机中的应用程序（进程）

“IP 地址+协议+端口”可以标识网络的进程，网络中进程通信即可实现。

什么是 Socket

Socket 英文愿意是“插孔”或“插座”，作为 BSD UNIX 的进程通信机制后，取后一种意思，通常也被称为**套接字**。使用 TCP/IP 协议的应用程序通常采用的**应用编程接**是 UNIX BSD 的套接字 Socket，来实现网络进程之间的通信。

Socket 用于描述 IP 地址和端口，是一个通信链的句柄，用来实现不同虚拟机或物理机之间的通信。应用程序通过 Socket 向网络发出请求或应答请求。网络中两个进程通过一个双向的通信连接实现数据的交换，建立网络通信连接至少需要一对 Socket，连接的一端称为一个 Socket。

具有唯一标识的网络进程可利用 Socket 进行通信，而 Socket 是在应用层和传输层之间的一个抽象层，Socket 把 TCP/IP 层复杂的操作抽象为简单的接口供应用层调用，以实现进程在网络中的通信。TCP/IP 协议存在于操作系统中，网络服务是通过操作系统提供的，因此在操作系统中增加支持 TCP/IP 的系统调用 Socket。

Socket 作为应用层与 TCP/IP 协议簇通信的中间软件抽象层，是一组接口。在设计模式中 Socket 其实就是一个**门面模式**，它将复杂的 TCP/IP 协议簇隐藏在 Socket 接口后面。对用户而言一组简单地接口就是全部，让 Socket 去组织数据已符合指定的协议。

Socket 抽象层

Socket 通信流程

Socket 可理解为一种特殊的文件，在服务器和客户端各自维护一个文件，并使用 SocketAPI 函数对其进行文件操作。在建立连接打开后，可以向各自文件写入内容供对方读取或读取对方内容，通信结束时关闭文件。在 UNIX 哲学中“一

切皆文件”，文件的操作模式基本为“打开-读写-关闭”三大步骤，Socket 其实就是这个模式的一个实现。

Socket 通信流程

服务器

1. 服务器根据 IP 地址类型（IPv4/IPv6)、Socket 类型和协议创建套接字
2. 服务端为 Socket 绑定 IP 地址和端口号
3. 服务端 Socket 监听端口请求，随时准备接收客户端发来的连接，此时 socket 并未被打开。

客户端

1. 客户端打开 Socket，根据服务器 IP 地址和端口试图连接服务端的 Socket。
2. 服务器 Socket 接收到客户端 Socket 请求，被动打开开始接收客户端请求，直到客户端返回连接信息，此时 Socket 进入阻塞状态。

交互过程

1. 客户端连接成功向服务端发送连接状态信息
2. 服务端 Accept 返回连接成功
3. 客户端向 Socket 写入数据
4. 服务端读取数据
5. 客户端关闭

伯克利 SocketAPI

历史

- Berkeley sockets 也称为 BSD Socket
- 1983 BSD Socket4.
- 1989 UNIX 均采用
- 2008 成为 POSIX 标准

头文件

- `sys/socket.h` 函数和数据结构定义
- `netinet/in.h` IPv4 和 IPv6xiangguan 协议
- `sys/un.h` UNIX 机器间通信
- `arpa/inet.h` 处理数字从操作系统字节序到网络字节序
- `netdb.h` 映射服务到 IP 地址

SocketAPI 函数

socket() 创建套接字，根据指定地址、数据类型、协议分配一个套接字的描述字及所用资源。

```
int socket(int domain, int type, int protocol)
```

参数：

- `domain`
  协议簇/域，通常为 AF_INET(IPv4)、AF_INET6(IPv6)
- `type`
  套接字类型，主要是 SOCK_STREAM(TCP)、SOCK_DGRAM(UDP)
- `protocol`
  通常为 0

Socket 有 3 种类型

- 流式 `SOCK_STREAM`
  流失套接字提供可靠的、面向连接的通信流，使用 TCP 协议从而保证数据传输的正确性和顺序性。
- 数据报 `SOCK_DGRAM`
  数据包套接字定义了一种无连接的服务，数据通过相互独立的报文进行传输，是无序的，不保证是可靠的无差错的，使用数据报协议 UDP。
- 原始 `SOCK_RAW`
  原始套接字允许对底层协议如 IP 或 ICMP 进行直接访问，功能强大但使用不便，主要用于协议开发。

返回值：
成功时返回非负整数

- `bind()` 绑定 socket 到 IP 地址和端口
- `listen()` 服务器监听客户端的连接
- `connect()` 客户端连接到服务器
- `accept()` 应用程序接收完成 3 次握手的客户端连接
- `send()`
- `recv()`
- `write()`
- `read()`
- `close()` 关闭 socket
- `gethostbyname()`IPv4 专用
- `gethostbyaddr()` IPv4 专用
- `select()`
- `poll()` 处理多个连接的读写和错误状态
- `getsocketopt()` 获得对应 socket 的选项值
- `setsocketopt()` 设置对应 socket 的选项值

Python socket 实现服务端和客户端数据传输（TCP）

本文链接：https://blog.csdn.net/weixin_44649870/article/details/87367670

- **服务器端**
- import socket
- #创建一个 socket 对象
- socket_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- host = "127.0.0.1"
- port = 9999
- #绑定地址
- socket_server.bind((host, port))
- #设置监听
- socket_server.listen(5)
- # socket_server.accept()返回一个元组，元素 1 为客户端的 socket 对象，元素 2 为客户端的地址(ip 地址，端口号)
- client_socket, address = socket_server.accept()
-
- #while 循环是为了让对话持续
- while True:
- #接收客户端的请求
- recvmsg = client_socket.recv(1024)
- #把接收到的数据进行解码
- strData = recvmsg.decode("utf-8")
- #设置退出条件
- if strData == 'q':
- break
- print("接收: %s" % strData)
- #输入
- msg = input("发送: ")
- #发送数据，需要进行编码
- client_socket.send(msg.encode("utf-8"))
- #关闭服务器端
- socket_server.close()

- **客户端**
- import socket
- #创建一个 socket 对象
- client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
-
- host = "127.0.0.1"

- port = 9999
- #连接服务端
- client.connect((host, port))
-
- while True:
- 	send_msg = input("发送: ")
- 	#设置退出条件
- 	if send_msg == "q":
- 		break
- 	send_msg = send_msg
- 	#发送数据，编码
- 	client.send(send_msg.encode("utf-8"))
- 	#接收服务端返回的数据
- 	msg = client.recv(1024)
- 	#解码
- 	print("接收：%s", % msg.decode("utf-8"))
- #关闭客户端
- client.close()

- **流程图**