# Python 全栈 - 001

腾蛇起陆　Python　全栈

---



---

# 0. 参考资料

http://www.cnblogs.com/linhaifeng/articles/5937962.html

# 1. 数据概述

机器数与真值

1. 机器数： 一个数在计算机中的二进制表示形式，叫做这个数的机器数。机器数是带符号的，在计算机用一个数的最高位存放符号，正数为 `0` ，负数为 `1` 。
2. 真值： 因为第一位是符号位，所以机器数的形式值就不等于真正的数字。所以，为了区别起见，将带符号位的机器数对应的真正数值称为机器数的真值。

原码、反码和补码

1. 原码就是符号位加上真值的绝对值，即用第一位表示符号，其余位表示值。
2. 反码：正数的反码是其本身。负数的反码是在其原码的基础上，符号位不变，其余各个位取反。
3. 补码：正数的补码是其本身。负数的补码是在其原码的基础上，符号位不变，其余各位取反，最后 `+1` （即在反码的基础上 `+1` ）。

在计算机系统中，数值一律用补码来表示（存储）。
主要原因：使用补码，可以将符号位和其他位统一处理；同时，减法也可按加法来处理。另外，两个用补码表示的数相加时，如果最高位（符号位）有进位，则进位被舍弃。

---

# 2. 网络基础

## 2.1 互联网的本质就是一系列的网络协议

世界上的所有计算机互相通信要有一个通用的标准。

## 2.2 `OSI` 七层协议

一系列的网络协议分成七层，从高到低：

- 应用层
- 表示层
- 会话层
- 传输层
- 网络层
- 数据链路层
- 物理层

五层分类：

- 应用层
- 传输层

- 网络层
- 数据链路层
- 物理层

四层分类：

- 应用层
- 传输层
- 网络层
- 网络接口层

---

## 2.2.1 物理层

物理层功能：主要是基于电器特性发送高低电压（电信号），高电压对应数字 `1` ，低电压对应数字 `0`
光缆、电缆、双绞线、无线电波等

---

## 2.2.2 数据链路层

数据链路层的由来：单纯的电信号 `0` 和 `1` 没有任何意义，必须规定电信号多少位一组，每组什么意思。
数据链路层的功能：定义了电信号的分组方式。

以太网协议
早期的时候各个公司都有自己的分组方式，后来形成了统一的标准，即以太网 `ethernet` 协议。

`ethernet` 规定

- 一组电信号构成一个数据包，或者叫数据帧
- 每一数据帧分成：报头 `head` 和数据 `data` 两部分。

| head | data |
| --- | --- |

`head` 包含：（固定 `18` 个字节）

- 发送者/源地址， 6 个字节 s_mac
- 接受者/目标地址， 6 个字节 d_mac
- 数据类型， 6 个字节

data 包含：（最短 46 字节，最长 1500 字节）

- 数据包的具体内容

head 长度 + data 长度 = 最短 64 字节，最长 1518 字节，超过最大限制就分片发送

**mac** 地址
head 中包含的源地址和目标地址的由来： ethernet 规定接入 internet 的设备都必须具备网卡，发送端和接收端的地址就是指网卡的地址，即 mac 地址。

**mac** 地址：每块网卡出厂时都被烧制上一个世界唯一的 mac 地址，是 12 位 16 进制数（即 48 位 2 进制数），前 6 位是厂商编号，后 6 位是流水线号。

> mac 地址又叫做物理地址。

广播
有了 mac 地址，同一网络内的两台主机就可以通信了。
（一台主机通用 arp 协议获取另外一台主机的 mac 地址）
ethernet 采用最原始的方式，广播进行通信。
比喻：广播就相当于在同一个屋子里，吼一嗓子我是谁，我要找谁，我要干什么。
同一网络中的计算机都会收到数据包，拆开后发现目标 mac 地址不是自己，就会丢弃，如果是自己，就会响应（还是以广播的方式）。

---

## 2.2.3 网络层

网络层由来：有了 ethernet 、 mac 地址和广播的发送方式，世界上的计算机就可以彼此通信了。
问题是世界范围的互联网是由一个个彼此隔离的小局域网组成的，如果所有的

通信都采用以太网的广播方式，那么一台机器发送的包全世界都会收到。
这就不仅仅是效率低的问题了，这会是一种灾难。

跨网络通信要把数据发给网关，由网关转发数据。
比喻：找其他教室的人要经过该教室的负责人，这个人就是网关。 IP 地址标识你在哪个教室， mac 地址标识你在该教室的哪个位置。

**IP** 协议

- 规定网络地址的协议叫 ip 协议，它定义的地址称之为 ip 地址，广泛采用的 v4 版本即 ipv4 ，它规定网络地址由 32 位 2 进制表示
- 范围 0.0.0.0 - 255.255.255.255
- 一个 ip 地址通常写成四段十进制数，例： 172.16.10.1

**IP** 数据包

- IP 数据包也分为 head 和 data 部分，无须为 IP 包定义单独的栏位，直接放入以太网包的 **data** 部分。

head ： 长度为 20 到 60 字节
data ： 最长为 65515 字节。
而以太网数据包的"数据"部分，最长只有 1500 字节。因此，如果 IP 数据包超过了 1500 字节，它就需要分割成几个以太网数据包，分开发送了。

**arp** 协议
arp 协议由来：计算机通信基本靠吼，即广播的方式，所有上层的包到最后都要封装上以太网头，然后通过以太网协议发送，在谈及以太网协议时候，我们了解到通信是基于 mac 的广播方式实现，计算机在发包时，获取自身的 mac 是容易的，如何获取目标主机的 mac ，就需要通过 arp 协议。

arp 协议功能：以广播的方式发送数据包，获取目标主机的 mac 地址。

| 源 **mac** | 目标 **mac** | 源 **IP** | 目标 **IP** | 数据部分 |
| --- | --- | --- | --- | --- |

| 发送端 mac | FF:FF:FF:FF:FF:FF | 172.16.10.10/24 | 172.16.10.11/24 | 数据 |
| --- | --- | --- | --- | --- |

> 目标 mac 全填 1 表示要获取目标 mac 地址。
> 假如不在同一网络，目标 IP 是 172.16.10.1 ，通过 arp 获取的是网关的 mac 地址

---

## 2.3.4 传输层

我们通过 IP 和 mac 找到了一台特定的主机，如何标识这台主机上的应用程序，答案就是端口，端口即应用程序与网卡关联的编号。
传输层功能：建立端口到端口的通信。
补充：端口范围 0 - 65535 ， 0 - 1023 为系统占用端口。

传输层的数据传输依靠 TCP 和 UDP 协议。
TCP 和 UDP 协议研究起来非常复杂，所以在上面封装了一个 socket 抽象层，开放一些可调用的方法，方便用户编程开发。

---

## 2.3.5 应用层

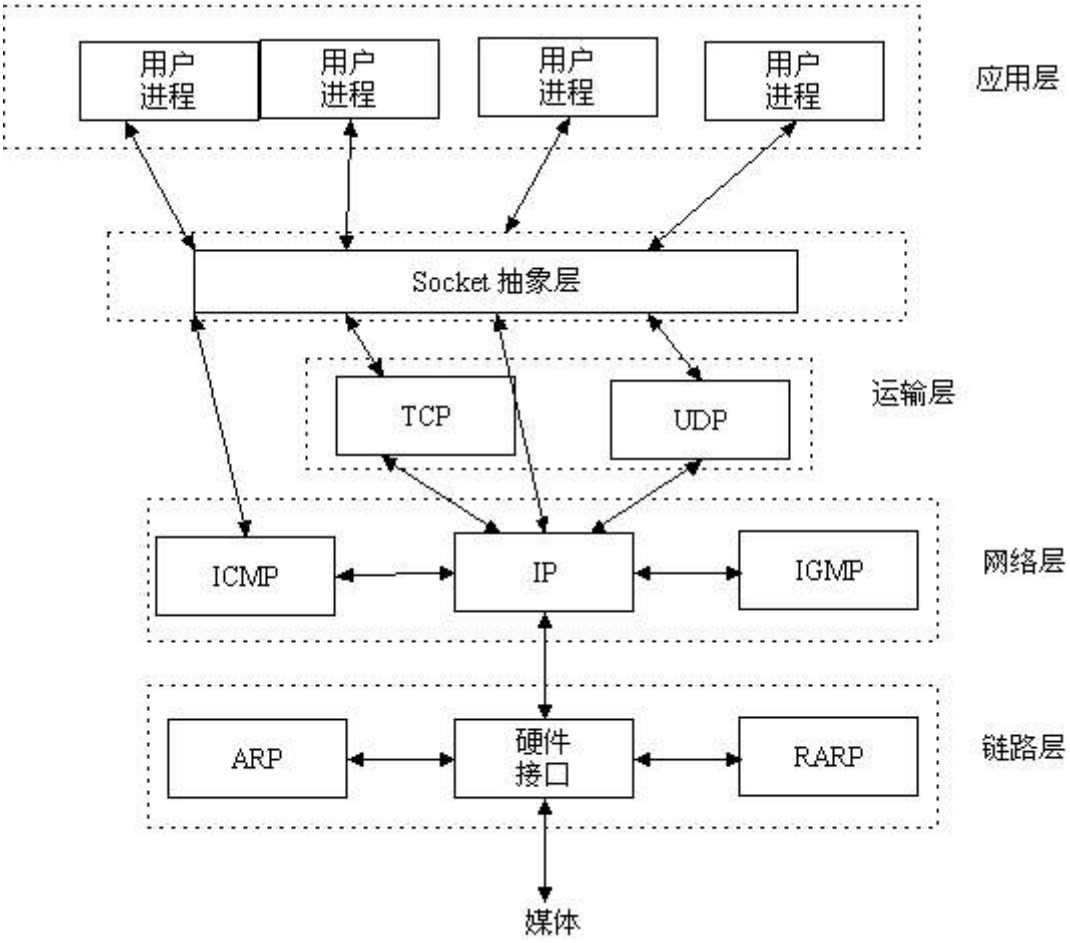
TCP 协议可以为各种各样的程序传递数据，比如 Email 、 WWW 、 FTP 等等。那么，必须有不同协议规定电子邮件、网页、 FTP 数据的格式，这些应用程序协议就构成了”应用层”。
应用层功能：规定应用程序的数据格式。
表示层：压缩、解压；加密解密。
会话层：建立会话。

发送数据是一个封装的过程，接受数据是一个解封装的过程。
封包自上而下封，拆包自下而上拆。

---

## 2.3.6 socket 套接字

socket 是在应用层和传输层之间的一个抽象层。
它把 TCP/IP 层复杂的操作抽象为几个简单的接口，供应用层调用，以实现进程在网络中通信。




---

2018.09.07