

happyGloria

2018年01月31日 阅读 1849

理解TCP/IP、UDP – 通过nodejs的net模块

1. 引子

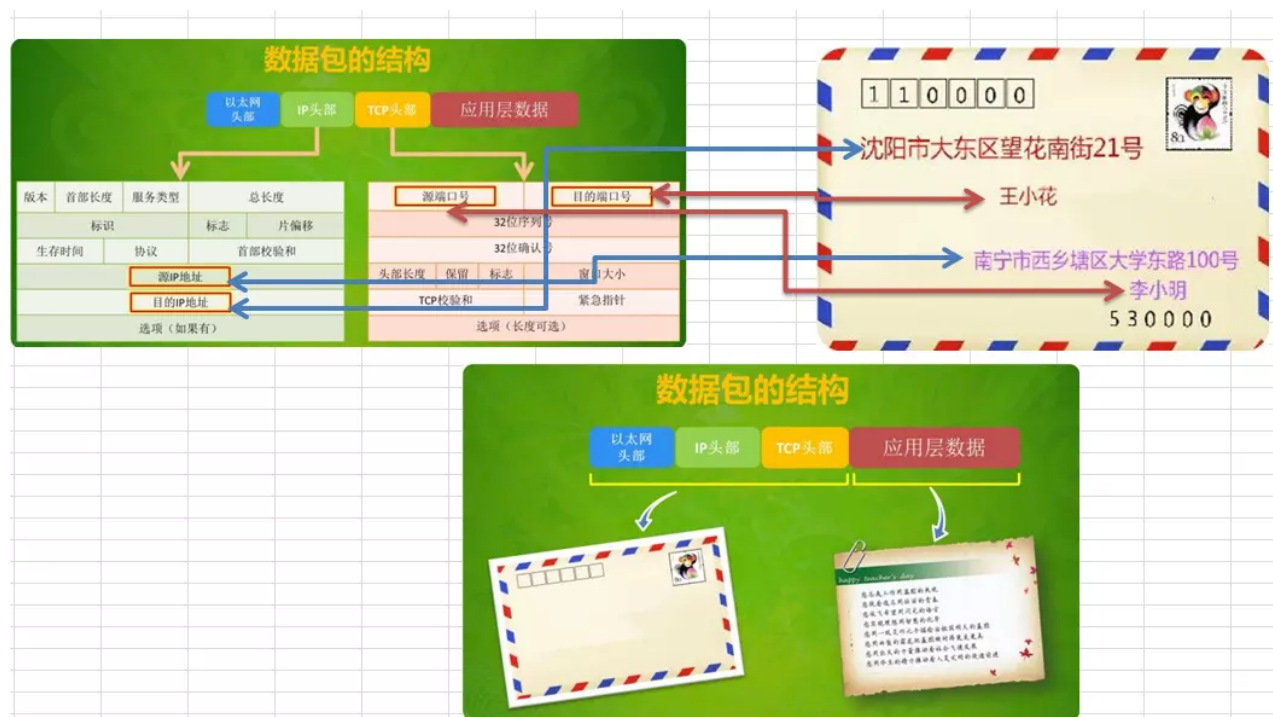
最近在学习node.js的net模块，涉及到了tcp/ip,arp,rcmp,http等协议,在这之前本人仅对http协议进行过深入的研究，至于其它协议仅仅只是知道有这些协议存在而已，未深入研究过。说实在的，网络协议概念很简单，但是也很抽象，网上查了很多资料，都是一些晦涩难懂的语言，所以个人觉得，明白协议的作用、怎么用、以何种形式用，再去看协议的具体工作过程更会让人印象深刻，下面就简明扼要的阐述TCP、IP、UDP。

TCP/IP协议，是传输控制协议/因特网互联协议，又名网络通讯协议，是Internet最基本的协议、Internet国际互联网络的基础，由网络层的IP协议和传输层的TCP协议组成。TCP/IP 定义了电子设备如何连入因特网，以及数据如何在它们之间传输的标准。协议采用了4层的层级结构，每一层都呼叫它的下一层所提供的协议来完成自己的需求。通俗而言：TCP负责发现传输的问题，一有问题就发出信号，要求重新传输，直到所有数据安全正确地传输到目的地。而IP则是给因特网的每一台联网设备规定一个地址。-- 百度百科

封面图中，展示了OSI七层及TCP/IP五层协议的对应关系；

- 网络由下往上分为物理层、数据链路层、网络层、传输层、应用层。
- IP协议对应于网络层，TCP协议对应于传输层，而HTTP协议对应于应用层，三者从本质上来说没有可比性，socket则是对TCP/IP协议的封装和应用（程序员层面上）。
- 也可以说，TCP/IP协议是传输层协议，主要解决数据如何在网络中传输，而HTTP是应用层协议，主要解决 如何包装数据。

2. 一张图让你了解TCP/IP到底是啥？

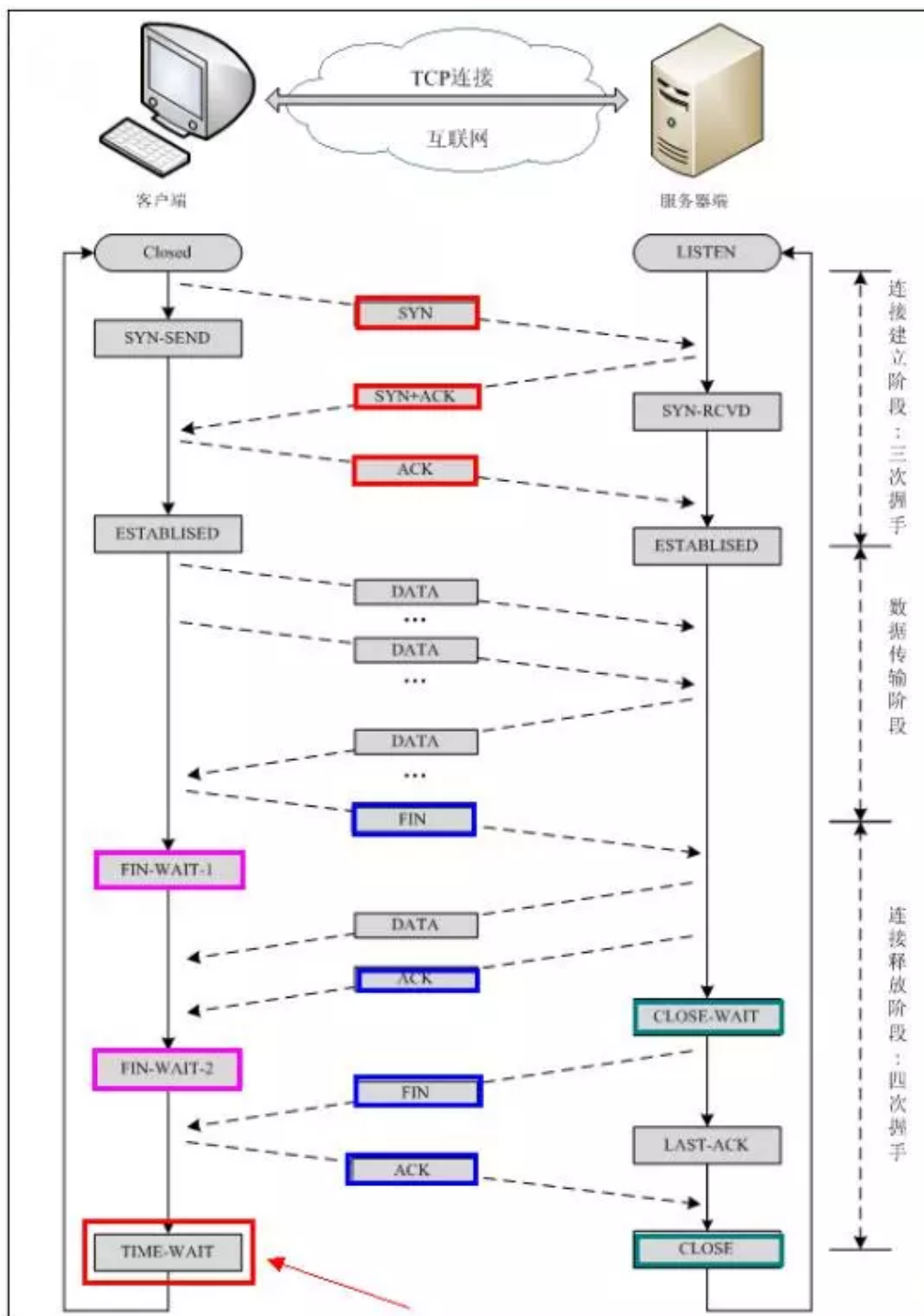


IP能锁定一台物理机器，对应着一张网卡，外界发来的数据包网卡都会接收。如果所有程序都需要监听网卡接收数据，每个包都被发到了所有应用程序，那应用程序符合不了，最后会垮掉，所以就诞生了端口这个标识，从数据安全层面考虑，一个标识号只能被一个应用程序监听。其实网卡都是被系统封装了，端口和进程之间的关系也是系统封装好的。我们只需要用socket就行，给定一个端口号就行了。其它的事都交给操作系统去做。TCP读取端口号，这个端口号就是创建socket时注册的，socket创建成功应该有一个process ID，这应该是操作系统来完成的，TCP于是就把[端口号 Process ID] 联系了起来，于是就和这个Process ID进程交换，完成数据的发送和接收。 [点击查](#)

[考链接](#)

通过这篇文章，我才知道IP和TCP到底做了些啥玩意..... 就是寻址和保证数据传递正确；

TCP三次握手四次断升



1. 三次握手

1. 第一次握手 主机A 通过一个标识为SYN标识位的数据段发送给主机B 请求连接，通过该数据段告诉主机B希望建立连接，需要B应答，并告诉主机B传输的起始序列号；



3. 第二次握手是 **主机A确认收到** 了主机B的数据段后可以开始传输实际数据。

2. 四次断开

1. **主机A** 发送FIN控制位发出 **断开连接的请求**；
2. **主机B** 进行响应，**确认收到断开连接请求**；
3. **主机B** 提出反方向的 **关闭要求**；
4. **主机A确认** 收到的主机B的 **关闭连接请求**；

问题：为什么断开要四次，而不是三次？因为主机B在响应收到断开连接请求的同时，还存在未发送完的数据；

4. UDP

UDP协议并不提供超时重传，出错重传等功能，所以说其是不可靠的协议。

5. TCP与UDP的区别

TCP(Transimision Control Protocal) ==> http ftp smtp ==> 电话

- 传输控制协议
- 可靠的、面向连接的协议
- 传输效率低

UDP(User Datagram Protocal) ==> qq, 微信 ==> 广播

- 用户数据报协议
- 不可靠的、无连接的服务
- 传输效率高

1. TCP是面向链接的，虽然说网络的不安全不稳定特性决定了多少次握手都不能保证连接的可靠性，但TCP的三次握手在最低限度上（实际上也很大程度保证了）保证了连接的可靠性；而UDP不是面向连接的，UDP传送数据前并不与对方建立连接，对接收到的数据也不发送确认信号，发送端不知道数据是否会正确接收，当然也不用重发，所以说UDP是无连接的、不可靠的传输协议。2. 也正由于1所说的特点，使得UDP的开销更小数据传输速率更高，因为TCP需要进行收发数据的确认，所以UDP的实时性更好。



对每个数据包进行编号然后由接收方进行验证啊什么的，即使是这样，UDP因为在底层协议的封装上没有采用类似TCP的“三次握手”而实现了TCP所无法达到的传输效率。

6. node.js net模块

- net模块也是node的核心模块,用于底层的网络通信;
- http.Server继承了net.Server;
- http客户端与http服务端的通信均依赖于socket (net.Socket) ;

6.1 net模块组成

主要包含两个部分:

1. **net.Server** tcp/server, 服务端TCP监听来自客户端的请求, 并使用TCP连接(socket)向客户端发送数据; 内部通过socket来实现与客户端的通信;
2. **net.Socket** tcp/本地, 客户端TCP连接到服务器, 并与服务器交换数据; socket的node实现, 实现了全双工的stream的接口;

6.2 服务端net.Server

```
let net = require('net')
let PORT = 8081
let HOST = 'localhost'
/**
 * 1. 创建一个TCP服务器实例, 调用listen函数开始监听指定端口;
 * 2. 传入net.createServer()的回调函数, 作为connection事件的处理函数;
 * 3. 在每个connection事件中, 该回调函数接收到的socket对象是唯一的;
 * 4. 该连接自动关联一个socket对象
 */
let server = net.createServer((socket) => {
  console.log('connection:' + socket.remoteAddress, socket.remotePort)
  // 为这个socket实例添加一个“data”事件处理函数
  socket.on('data', (data) => {
    console.log('DATA' + socket.remoteAddress + ":" + data);
    socket.write('You said "' + data + '"\r\n') // 向客户端回发该数据
  })

  socket.on('end', () => {
    console.log('客户端关闭')
  })
})
/**
```

```
        * */
        server.unref();
        //调用了该方法，则所有的客户端关闭跟本服务器的连接后，将关闭服务器
    })

    // 客户端关闭事件
    socket.on('close', () => {
        console.log('CLOSED: ' + socket.remoteAddress + ' ' + socket.remotePort);
    })

    /*socket.pause()
    socket.setTimeout(3000) //设置客户端超时时间，如果客户端一直不输入，超过这个时间，就认为超时了
    socket.on('timeout', () => {
        console.log('超时了')
        socket.pipe(ws, {end: false})
        // 默认情况下，当可读流读到末尾的时候会关闭可写流
    })*
    })

    server.listen(PORT, HOST, () => {
        console.log('服务端的地址是: ', server.address())
    })

    server.on('error', (err) => {
        console.log(err)
    })

    //服务端也可以通过显式处理"connection"事件来建立TCP连接，只是写法不同，二者没有区别即：
    /*
    let server = net.createServer()
    server.listen(PORT,HOST)
    server.on('connection', (socket) => {
        console.log('CONNECTED: ' + sock.remoteAddress + ':' + sock.remotePort);
    })*
    server.on('close', () => {
        //关闭服务器，停止接收新的客户端的请求
        console.log('close事件: 服务端关闭 ');
    })

    server.on('error', (error) => {
        console.log('error事件: 服务端异常: ' + error.message );
    })
```

6.3 客户端net.Socket



//创建一个TCP客户端连接到刚创建的服务器上，该客户端向服务器发送一串消息，并在得到服务器的反馈后关闭连接。

```
var client = new net.Socket()
let PORT = 8081
let HOST = 'localhost'

client.connect(PORT, HOST, () => {
  console.log('connect to ' + HOST + ':' + PORT)
  client.write('I am happyGloria.') //建立连接后立即向服务器发送数据，服务器将收到这些数据
})

client.on('data', (data) => {
  console.log('DATA: ' + data)
  client.destroy() // 完全关闭连接
})

client.on('close', function () {
  console.log('Connection closed')
})
```

6.4 基于tcp的聊天室

```
let net = require('net')
let util = require('util')
let HOST = 'localhost'
let PORT = 8082
let clients = {}

function broadcast (username, msg) {
  for (let name in clients) {
    if (name !== username) {
      clients[name].write(msg + '\r\n')
    }
  }
}

let server = net.createServer((socket) => {
  socket.setEncoding('utf8')
  server.getConnections((err, count) => {
    socket.write('在线人数是' + count + '位，请输入你的昵称:\r\n')
  })

  let username
  socket.on('data', (data) => {
```



```
        } else {
          if (clients[data]) {
            socket.write('您的昵称' + data + '被占用了, 请您更换新的昵称\r\n')
          } else {
            username = data
            clients[username] = socket
            broadcast(username, `欢迎${username}加入`)
          }
        }
      })

      socket.on('end', () => {
        broadcast(username, `${username}离开聊天室`)
        clients[username] && clients[username].destroy()
        delete clients[username]
      })
    })

    server.listen(PORT, HOST, () => {
      console.log(`tcp聊天室已启动, 地址是${util.inspect(server.address())}`)
    })
  })
}
```

参考资料: <https://www.zhihu.com/question/51074319>

关注下面的标签, 发现更多相似文章

Node.js

安装掘金浏览器插件

打开新标签页发现好内容, 掘金、GitHub、Dribbble、ProductHunt 等站点内容轻松获取。快来安装掘金浏览器插件获取高质量内容吧!



评论

输入评论...

SZOUc

四次分手个人认为有点误解, tcp是全双工, 形象点就是有两根线, A发到B收, B发到A收, 二次分手只断开一根线, 而且互不相关, 并不是断开同时还有数据。上课时给学生举例, 女孩对男孩说, 我们分手



9月前

 回复

SZOUc

四次分手个人认为有点误解，tcp是全双工，形象点就是有两根线，A发到B收，B发到A收，二次分手只能断开一根线，而且互不相关，并不是断开同时还有数据。上课时给学生举例，女孩对男孩说，我们分手吧。男孩，嗯，知道了。此时女孩方面是分手了，但是直到男孩说，我们分手吧，女孩确认后才真正的断开了。



9月前

 回复

Harpia King

精彩


9月前

 回复

相关推荐

专栏 · 黑金团队 · 2小时前 · Electron / Node.js

基于Electron + nodejs + 小程序 实现弹幕小工具（上篇）

 15



专栏 · BlameDeng · 21小时前 · Node.js / Vue.js

「伪全栈」Vue+Node搭建一个商城应用

 59

 19

热 · 专栏 · BiaoChenXuYing · 1天前 · React.js

react + node + express + ant + mongodb 的简洁兼时尚的博客网站

 144

 20

TesterNo1 · 22小时前 · Google / Node.js / Chrome

谷歌发布新的 Node.js Web 渲染界面 Carlo


 18



专栏 · 沈赟杰 · 1天前 · NPM / 前端

CPM – 轻量的NPM私有源程序搭建

 69

 12

专栏 · 黑金团队 · 1天前 · 前端 / Node.js

基于Electron + nodejs + 小程序 实现弹幕小工具（开篇）

热 · 专栏 · 黑金团队 · 2天前 · Node.js

WEB 前端模块化都有什么？

 82

 12

专栏 · 创宇前端 · 1天前 · 数据库 / Elasticsearch


从安装到入门：ElasticSearch 快速学习手册


 30

 2

热 · 专栏 · 殷荣桢 · 8天前 · React.js

看看这些被同事喷的JS代码风格你写过多少

 708

 146

热 · 专栏 · ixlei · 9天前 · Node.js

浅谈前端错误处理

 89

 2

