

Node.js API

```
http http.createServer(function(req,res){}) NodeJS 1. exports. = 2. module.exports =
= require('js ') fs 1. fs.readFile(" ",function(err,data){}) 1. __dirname 2. __filename
path.join() 2. path.resolve() 3. path.extname() mime ( ) 1. mime.getType() : res.writeHe
"Content-type": mime.getType('.jpg') }); url //
http://www.yts.com/api/index.html?username=rose&type=flower 1. const result = url.parse(url ) 2
result.pathname ---> api/index.html 3. result.query ----> "username=rose&type=flower"
```

- [HTTP](#)
- [NET](#)

HTTP

Node.js

http

1. http
2. http.createServer()
3. response.end()
4. listen

```
//  
var http = require('http');  
  
http.createServer(function(request, response){  
  response.end(' Hello Node ');  
}).listen(8080);
```

-- GET

GET

request.method

request.url

```
var http = require('http');  
var url = require('url');  
  
http.createServer(function(req, res){  
  var params = url.parse(req.url, true).query;  
  res.end(params);  
  
}).listen(3000);
```

-- POST

GET

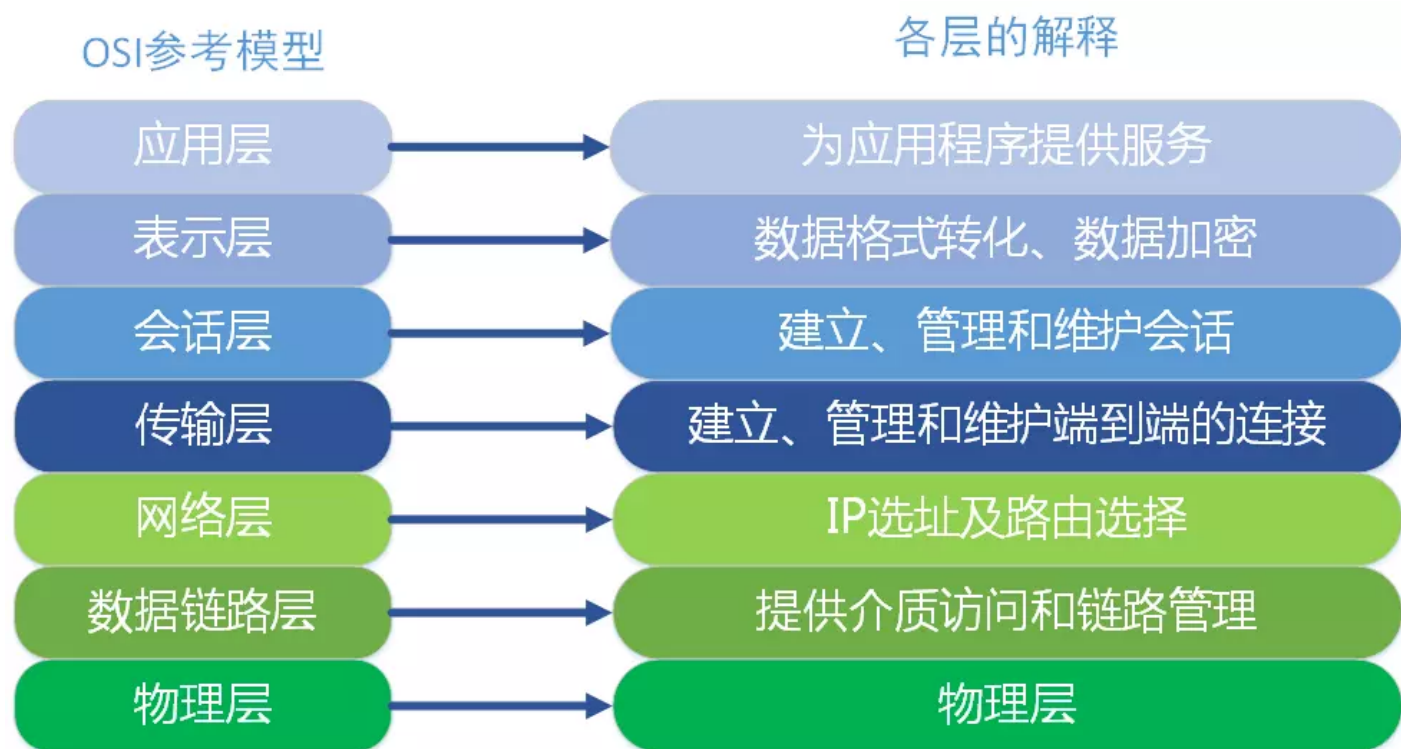
POST

url

```
var http = require('http');  
var util = require('util');  
var querystring = require('querystring');
```

```
http.createServer( function(req, res){  
  //      post  
  var post = '';  
  
  //  req data          post  
  req.on(' data', function(chunk){  
    post += chunk;  
  });  
  
  //  end      querystring.parse post      POST  
  req.on(' end', function(){  
    post = querystring.parse(post);  
    res.end(util.inspect(post));  
  });  
}).listen(3000);
```

NET



TCP/IP

第7层 应用层

各种应用程序协议，如 HTTP、FTP、SMTP、POP3。



7

第6层 表示层

信息的语法语义以及它们的关联，如加密解密、转换翻译、压缩解压缩。

6

第5层 会话层

不同机器上的用户之间建立及管理会话。

5

第4层 传输层

接受上一层的数据，在必要的时候把数据进行分割，并将这些数据交给网络层，且保证这些数据段有效到达对端。

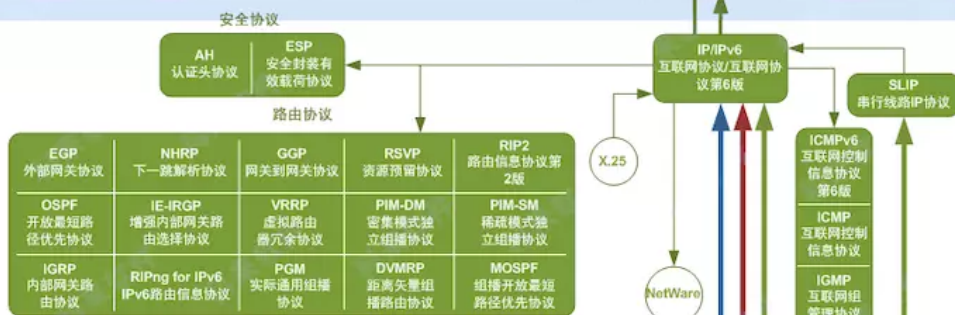
4

TCP 传输控制协议
UDP 用户数据报协议

第3层 网络层

控制子网的运行，如逻辑编址、分组传输、路由选择。

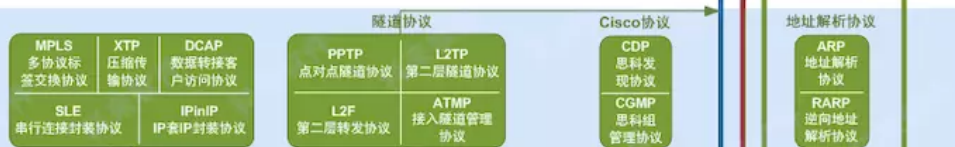
3



第2层 数据链路层

物理寻址，同时将原始比特流转变为逻辑传输线路。

2



第1层 物理层

机械、电子、定时接口通信信道上的原始比特流传输。

1

IEEE 802.2
Ethernet v.2
Internetwork

net dgram http https

Node

net	TCP
dgram	UDP
http	HTTP
https	HTTPS

- TCP 7
- UDP

- IP IP IP
- IP
- IP ICMP
- ARP IP MAC MAC
- IP

0 1

url

url

3

- - SYN - -
- - SYN/ACK - -
- - ACK - -

4

- - FIN
- - FIN ACK 1 SYN FIN
- - FIN
- - ACK 1

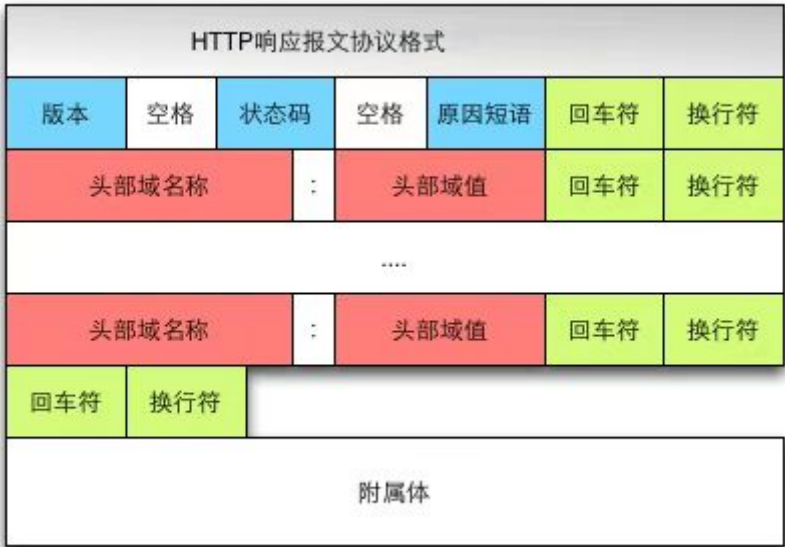
IP
DNS -> -> ...
web HTTP
http://example.com http://www.example.com

HTTP
HTML
HTML CSS JS

1XX	2XX	3XX	4XX	5XX
	200 OK	301	400	500
	204	302 ()	401 HTTP	503 307 302
	206	303 URL GET	403	
		304 GET	404	

http/https

http | TCP/IP | https | TLS SSL | http/https | TCP | http | http |



```
HTTP/1.0 200 OK    //
```

```
Content-type: text/plain    //
```

```
Content-length: 19        //
```

```
Hi I'm a message!    //
```

Node`http` `net`

API

TCP UDP

Node `net` TCP API

`net`

```
var net = require("net")
```

<code>net.createServer([options][, connectionListener])</code>	TCP	connectionListener	'connection'			
<code>net.connect(options[, connectionListener])</code>	'net.Socket'	socket		'connect'		
<code>net.createConnection(options[, connectionListener])</code>	port	host	TCP	host	'localhost'	
<code>net.connect(port[, host][, connectListener])</code>	port	host	TCP	host	'localhost'	connect
	'connect'	'net.Socket'				

<code>net.createConnection(port[, host][, connectListener])</code>	port host TCP host 'localhost'connectListener 'net.Socket'
<code>net.connect(path[, connectListener])</code>	path unix socket connectListener 'connectListener'
<code>net.createConnection(path[, connectListener])</code>	path unix socket connectListener 'connectListener'
<code>net.isIP(input)</code>	IP IPV4 4 IPV6 6 0
<code>net.isIPv4(input)</code>	IPV4 true false
<code>net.isIPv6(input)</code>	IPV6 true false

- `net.Socket` TCP UNIX Socket
- `net.createServer`
- `net.Socket`||`net.connect`

net.Server

`net.Server` || TCP

<code>server.listen(port[, host][, backlog][, callback])</code>	port host ac host IPv4 (INADDR_ANY)
<code>server.listen(path[, callback])</code>	path socket
<code>server.listen(handle[, callback])</code>	
<code>server.listen(options[, callback])</code>	options port, host, backlog, callback , [host], [backlog], [callback] path UNIX socket
<code>server.close([callback])</code>	'close'
<code>server.address()</code>	
<code>server.unref()</code>	unref
<code>server.ref()</code>	unref unref ref
<code>server.getConnections(callback)</code>	socket 2 err count

```
let server = net.createServer((socket) => {});
server.listen(3000, () => {});
```

net.Socket

`net.Socket` TCP UNIX Socket `net.Socket` (`connect()` , `Node`

- listening server.listen
- connection socket net.Socket
- close

lookup	UNIX sokcet
connect	socket
data	
end	socket FIN
timeout	socket socket
drain	
error	
close	socket had_error socket

```

let server = net.createServer((socket) => {
  socket.on('data', (data) => {});
  socket.on('end', () => {});
  socket.on('error', (err) => {});
  socket.on('close', () => {});
});
server.on('close', (socket) => {});
server.on('error', (e) => {});

```

net.Sockets

net.Socket || socket

socket.connect(path[, connectListener])	unix socket net.createConnection socket
socket.setEncoding([encoding])	
socket.write(data[, encoding][, callback])	socket UTF8
socket.end([data][, encoding])	socket FIN
socket.destroy()	I/O
socket.pause()	data
socket.resume()	pause()
socket.setTimeout(timeout[, callback])	socket timeout socket
socket.setNoDelay([noDelay])	Nagle TCP noDelay tr

<code>socket.setKeepAlive([enable][, initialDelay])</code>	/	socket	probe	false
<code>socket.address()</code>		3	{ port: 12346, family: 'l	
<code>socket.unref()</code>		unref	unref	unref
<code>socket.ref()</code>	unref	unref	ref	

`new net.Socket([options])` || `socket`

```
let server = net.createServer((socket) => {
  socket.setEncoding('utf8');
  socket.write();
  socket.end();
});
```

TCP HTTP

net http

`socket.pipe` || `fs.createWriteStream` || `message.txt`

`http://localhost:3000` hello

```
let net = require('net');
let server = net.createServer({
  // pauseOnConnect true,
  pauseOnConnect: true
}, (socket) => {
  socket.setEncoding('utf8');
  socket.on('data', (data) => {
    console.log(data);
  });
  socket.on('end', () => {
    console.log('client disconnected');
  });
  //
  socket.on('error', (err) => {
    console.log("error");
  });
  socket.on('close', () => {
    console.log("close socket");
  });
});
```

```

    });
    socket.end(`
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 5

hello`)
    console.log('request');
  });

server.listen(3000, () => {
  console.log('opened server on', server.address());
});
server.on('connection', (socket) => {
  console.log('connection');
});

//server.unref(); // node server
//
server.on('close', (socket) => {
  console.log('close server');
});
server.on('error', (e) => {
  if (e.code === 'EADDRINUSE') {
    console.log('Address in use, retrying...');
    setTimeout(() => {
      server.close();
      server.listen(PORT, HOST);
    }, 1000);
  }
});

```

postman || message.txt

```

POST /abc HTTP/1.1
Content-Type: multipart/form-data; boundary=-----879095998142409176007484
abc: 123
bbb: ccc
ddd: eee
token:

```

```
eyJkYXRhIjp7ImducHV0RWlhaWwiOiJsZWlubiIsImducHV0UGFzc3dvcmQiOiIxMjMifSwiY3JlYXRLZCI6MTU0NzA0MTE  
cache-control: no-cache  
Postman-Token: 97b4950a-1169-407b-8787-ab238d3954d4  
User-Agent: PostmanRuntime/7.6.0  
Accept: */*  
Host: localhost:3000  
cookie: csrfToken=58RWJaRa3ZuA2uIp7cxn34pC  
accept-encoding: gzip, deflate  
content-length: 157  
Connection: keep-alive  
  
-----879095998142409176007484  
Content-Disposition: form-data; name="x"  
  
x  
  
-----879095998142409176007484--
```

net|http|net|

```
var statusLine = `HTTP/1.1 ${statusCode} ${this.statusMessage}${CRLF}`; // line 252  
  
function Server(options, requestListener) {  
  net.Server.call(this, { allowHalfOpen: true });  
  if (requestListener) {  
    this.on('request', requestListener);  
  }  
} // line 283  
  
net.Server.call(this, { allowHalfOpen: true }); //line 298
```

- [Node](#)
- [tcp http](#)