http-protocol

HTTP协议

http请求和tcp链接不是一个概念，在一个tcp链接里，可以发送多个http请求，之前的协议版本是不可以这么做的，从http/1.1里面就可以这样做了，tcp链接对应的是多个http请求，而一个http请求肯定是在某个tcp链接里面进行发送的。

5层网络模型介绍

应用层

构建于TCP协议之上，为应用软件提供服务，应用层也是最高的一层直接面向用户。

- www万维网
- FTP文件传输协议
- DNS协议: 域名与IP的转换
- 邮件传输
- DCHP协议

传输层

传输层向用户提供可靠的端到端(End-to-End)服务，主要有两个协议分别是TCP、 UDP协议， 大多数情况下我们使用的是TCP协议，它是一个更可靠的数据传输协议。

- 协议对比TCP
    - 面向链接: 需要对方主机在线，并建立链接。
    - 面向字节流: 你给我一堆字节流的数据，我给你发送出去，但是每次发送多少是我说了算，每次选出一段字节发送的时候，都会带上一个序号，这个序号就是发送的这段字节中编号最小的字节的编号。
    - 可靠: 保证数据有序的到达对方主机，每发送一个数据就会期待收到对方的回复，如果在指定时间内收到了对方的回复，就确认为数据到达，如果超过一定时间没收到对方回复，就认为对方没收到，在重新发送一遍。
- 协议对比UDP
    - 面向无链接: 发送的时候不关心对方主机在线，可以离线发送。
    - 面向报文: 一次发送一段数据。
    - 不可靠: 只负责发送出去，至于接收方有没有收到就不管了。

网络层

数据链路层

物理层

http协议发展历史

http/0.9

只有一个命令GET，对应我们现在的请求GET、POST，没有header等描述数据的信息，服务器发送完毕数据就关闭TCP链接，每个http请求都要经历一次dns域名解析、传输和四次挥手，这样反复创建和断开tcp链接的开销是巨大的，在现在看来这种方式很糟糕。

http/1.0

- 增加了很多命令POST、GET、HEAD
- 等增status code和header

  status code描述服务端处理某一个请求之后它的状态， header是不管发送还是请求一个数据它的描述。
- 多字符集支持、多部分发送、权限、缓存等。

http/1.1

- 持久链接
- 管道机制(pipeline)

  可以在同一个链接里发送多个请求，但是在服务端对于进来的请求都是要按照顺序进行内容的返回，如果前一个请求很慢，后一个请求很多，它也需要第一个请求发送之后，后一个请求才可以发送，这块在http2里面进行了优化
- 增加host和其他功能

  增加host可以在同一台物理服务器上跑多个web服务，例如一个nodejs的web服务，一个java的web服务

http/2

- 所有数据以二进制传输
- 同一个链接里面发送多个请求，不在需要按照顺序来
- 头信息压缩以及推送等提高效率的功能

http三次握手

先清楚一个概念http请求与tcp链接之间的关系，在客户端向服务端请求和返回的过程中，是需要去创建一个TCP connection，因为http是不存在链接这样一个概念的，它只有请求和响应这样一个概念，请求和响应都是一个数据包，中间要通过一个传输通道，这个传输通道就是在TCP里面创建了一个从客户端发起和服务端接收的一个链接，TCP链接在创建的时候是有一个三次握手(三次网络传输)这样一个消耗在的。

三次握手时序图

至于为什么要经过三次握手呢，是为了防止服务端开启一些无用的链接，网络传输是有延时的，中间可能隔着非常远的距离，通过光纤或者中间代理服务器等，客户端发送一个请求，服务端收到之后如果直接创建一个链接，返回内容给到客户端，因为网络传输原因，这个数据包丢失了，客户端就一直接收不到服务器返回的这个数据，超过了客户端设置的时间就关闭了，那么这时候服务端是不知道的，它的端口就会开着等待客户端发送实际的请求数据，服务这个开销也就浪费掉了。

三次握手数据包详细内容分析

这里采用的是wireshark 官网地址 https://www.wireshark.org/，是一个很好的网络数据包抓取和分析软件。

// todo

URI/URL/URN

URI

Uniform Resource Identifier/统一资源标志符，用来标示互联网上唯一的信息资源，包括URL和URN。

URL

Uniform Resource Locator/统一资源定位器

URN

永久统一资源定位符，例如资源被移动后如果是URL则会返回404，在URN中资源被移动之后还能被找到，当前还没有什么成熟的使用方案

跨域CORS

关于浏览器跨域的原理，一个请求在浏览器端发送出去后，是会收到返回值响应的，只不过浏览器在解析这个请求的响应之后，发现不属于浏览器的同源策略(地址里面的协议、域名和端口号均相同)，不允许进行拦截。如果是在curl里面发送一个请求，都是没有跨域这样一个概念的，下面是例子进行分析：

server.html

在这个html里面采用ajax请求3011这个服务

```
<html>
    <head>
        <meta charset="utf-8" />
        <title>cors</title>
    </head>
    <body>
        <script>
            const xhr = new XMLHttpRequest();

            xhr.open('GET', 'http://127.0.0.1:3011/');
            xhr.send();
        </script>
    </body>
</html>
```

server.3011.js

```
const http = require('http');
const port = 3011;

http.createServer((request, response) => {
    console.log('request url: ', request.url);

    response.end('3011 port serve for you');
}).listen(port);

console.log('server listening on port ', port);
```

server.3010.js

对上面定义的server.html模版进行渲染，从而在该模版里调用3011这个端口服务

```
const http = require('http');
const fs = require('fs');
const port = 3010;

http.createServer((request, response) => {
    console.log('request url: ', request.url);

    const html = fs.readFileSync('server.html', 'utf-8');

    response.writeHead(200, {
        'Content-Type': 'text/html',
    });
    response.end(html);

}).listen(port);

console.log('server listening on port ', port);
```

打开浏览器，地址栏输入http://127.0.0.1:3010/，会看到以下提示not allowed access，但是server.3011.js，会打印出 `request url: /`，说明浏览器在发送这个请求的时候并不知道服务是不是跨域的，还是会发送请求并接收服务端返回的内容，但是当浏览器接收到响应后在headers里面没有看到 `Access-Control-Allow-Origin:` 被设置为允许，会把这个请求返回的内容给忽略掉，并且在命令行报下面的错误。

```
Failed to load http://127.0.0.1:3011/: No 'Access-Control-Allow-Origin'
header is present on the requested resource. Origin
'http://127.0.0.1:3010' is therefore not allowed access.
```

解决的几种方法

- 设置Access-Control-Allow-Origin
    - 设置为*表示，可以接收任意域名的访问

```
http.createServer((request, response) => {

    response.writeHead(200, {
        'Access-Control-Allow-Origin': '*'
    })
}).listen(port);
```

    - 也可以设置为特定域名访问

```
http.createServer((request, response) => {

    response.writeHead(200, {
        'Access-Control-Allow-Origin': 'http://127.0.0.1:3010/'
    })
}).listen(port);
```

    - 如果有多个域名访问可以在服务端动态设置

```
http.createServer((request, response) => {
    const origin = request.headers.origin;

    if ([
        'http://127.0.0.1:3010'
    ].indexOf(origin) !== -1) {
        response.writeHead(200, {
            'Access-Control-Allow-Origin': origin,
        })
    }
}).listen(port);
```

- jsonp

浏览器是允许像link、img、script标签在路径上加载一些内容进行请求时是允许跨域的，那么jsonp的实现原理就是在script标签里面加载了一个链接，去访问服务器的某个请求，返回内容

```
<html>
    <head>
        <meta charset="utf-8" />
        <title>cors</title>
    </head>
    <body>
        <!-- <script>
```

```
            const xhr = new XMLHttpRequest();

            xhr.open('GET', 'http://127.0.0.1:3011/');
            xhr.send();
        </script> -->

        <script src="http://127.0.0.1:3011/"></script>
    </body>
</html>
```