

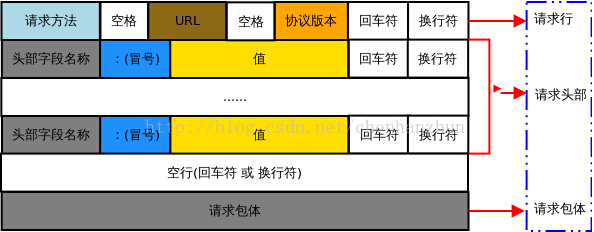
ajax要自己解決跨域問題

**http:**

**請求報文格式（請求行、請求頭、空行 和 請求包體）**

**響應報文格式（狀態行、響應頭部、空行 和 響應包體）**

https://www.cnblogs.com/breka/p/11635540.html



請求行：請求行由方法字段、URL 字段 和HTTP 協議版本字段 3 個部分組成，他們之間使用空格隔開。常用的 HTTP 請求方法有 GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT;

 GET：當客戶端要從服務器中讀取某個資源時，使用GET 方法。GET 方法要求服務器將URL 定位的資源放在響應報文的部分，回[送給](http://www.chinabyte.com/keyword/%E9%80%81%E7%BB%99/)客戶端，即向服務器請求某個資源。使用GET 方法時，請求參數和對應的值附加在 URL 後面，利用一個問號(“?”)代表URL 的結尾與請求參數的開始，傳遞參數長度受限制。例如，/index.jsp?id=100&op=bind。

　　● POST：當客戶端給服務器提供信息較多時可以使用POST 方法，POST 方法向服務器提交數據，比如完成表單數據的提交，將數據提交給服務器處理。GET 一般用於獲取/查詢資源信息，POST 會附帶用戶數據，一般用於更新資源信息。POST 方法將請求參數[封裝](http://www.chinabyte.com/keyword/%E5%B0%81%E8%A3%85/)在HTTP 請求數據中，以名稱/值的形式出現，可以傳輸大量數據;

　　請求頭部：請求頭部由關鍵字/值對組成，每行一對，關鍵字和值用英文冒號“:”分隔。請求頭部通知服務器有關於客戶端請求的信息，典型的請求頭有：

　　● User-Agent：產生請求的[瀏覽器](http://www.chinabyte.com/keyword/%E6%B5%8F%E8%A7%88%E5%99%A8/)類型;

　　● Accept：客戶端可識別的響應內容類型列表;星號 “ \* ” 用於按范圍將類型分組，用 “ \*/\* ” 指示可接受全部類型，用“ type/\* ”指示可接受 type 類型的所有子類型;

　　● Accept-Language：客戶端可接受的自然語言;

　　● Accept-Encoding：客戶端可接受的編碼壓縮格式;

　　● Accept-Charset：可接受的應答的字符集;

　　● Host：請求的主機名，允許多個[域名](http://www.chinabyte.com/keyword/%E5%9F%9F%E5%90%8D/)同處一個IP 地址，即虛擬主機;

　　● connection：連接方式(close 或 keepalive);

　　● Cookie：[存儲](http://storage.chinabyte.com/" \t "_blank)於客戶端擴展字段，向同一域名的服務端發送屬於該域的cookie;

空行：最後一個請求頭之後是一個空行，發送回車符和換行符，通知服務器以下不再有請求頭;

　　請求包體：請求包體不在 GET 方法中使用，而是在POST 方法中使用。POST 方法適用於需要客戶填寫表單的場合。與請求包體相關的最常使用的是包體類型 Content-Type 和包體長度 Content-Length;



123請求行

4請求頭

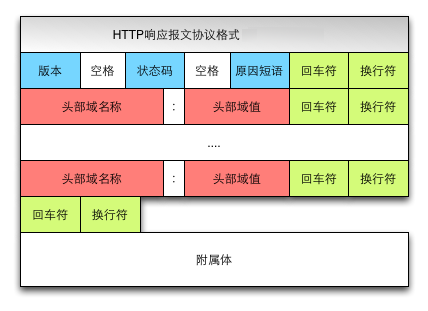
4,5之間 空行

5 請求體(post才有)



**HTTP 響應報文**

http響應報文由狀態行、響應頭部、空行 和 響應包體 4 個部分組成，如下圖所示：



下面對響應報文格式進行簡單的分析：

　　狀態行：狀態行由 HTTP 協議版本字段、狀態碼和狀態碼的描述文本 3 個部分組成，他們之間使用空格隔開;

　　● 狀態碼由三位數字組成，第一位數字表示響應的類型，常用的狀態碼有五大類如下所示：

　　1xx：表示服務器已接收了客戶端請求，客戶端可繼續發送請求;

　　2xx：表示服務器已成功接收到請求並進行處理;

　　3xx：表示服務器要求客戶端重定向;

　　4xx：表示客戶端的請求有非法內容;

　　5xx：表示服務器未能正常處理客戶端的請求而出現意外錯誤;

　　● 狀態碼描述文本有如下取值：

　　200 OK：表示客戶端請求成功;

　　400 Bad Request：表示客戶端請求有語法錯誤，不能被服務器所理解;

　　401 Unauthonzed：表示請求未經授權，該狀態代碼必須與 WWW-Authenticate 報頭域一起使用;

　　403 Forbidden：表示服務器收到請求，但是拒絕提供服務，通常會在響應正文中給出不提供服務的原因;

　　404 Not Found：請求的資源不存在，例如，輸入了錯誤的URL;

　　500 Internal Server Error：表示服務器發生不可預期的錯誤，導致無法完成客戶端的請求;

　　503 Service Unavailable：表示服務器當前不能夠處理客戶端的請求，在一段時間之後，服務器可能會恢復正常;

　　響應頭部：響應頭可能包括：

　　Location：Location響應報頭域用於重定向接受者到一個新的位置。例如：客戶端所請求的頁面已不存在原先的位置，為了讓客戶端重定向到這個頁面新的位置，服務器端可以發回Location響應報頭後使用重定向語句，讓客戶端去訪問新的域名所對應的服務器上的資源;

　　Server：Server 響應報頭域包含了服務器用來處理請求的軟件信息及其版本。它和 User-Agent 請求報頭域是相對應的，前者發送服務器端軟件的信息，後者發送客戶端軟件(瀏覽器)和[操作系統](http://soft.chinabyte.com/os/" \t "_blank)的信息。

　　Vary：指示不可緩存的請求頭列表;

　　Connection：連接方式;

　　對於請求來說：close(告訴[WEB 服務器](http://www.chinabyte.com/keyword/web%E6%9C%8D%E5%8A%A1%E5%99%A8/" \t "_blank)或者[代理服務器](http://www.chinabyte.com/keyword/%E4%BB%A3%E7%90%86%E6%9C%8D%E5%8A%A1%E5%99%A8/)，在完成本次請求的響應後，斷開連接，不等待本次連接的後續請求了)。keepalive(告訴WEB服務器或者代理服務器，在完成本次請求的響應後，保持連接，等待本次連接的後續請求);

　　對於響應來說：close(連接已經關閉); keepalive(連接保持著，在等待本次連接的後續請求); Keep-Alive：如果瀏覽器請求保持連接，則該頭部表明希望WEB 服務器保持連接多長時間(秒);例如：Keep-Alive：300;

　　WWW-Authenticate：WWW-Authenticate響應報頭域必須被包含在401 (未授權的)響應消息中，這個報頭域和前面講到的Authorization 請求報頭域是相關的，當客戶端收到 401 響應消息，就要決定是否請求服務器對其進行驗證。如果要求服務器對其進行驗證，就可以發送一個包含了Authorization 報頭域的請求;

　　空行：最後一個響應頭部之後是一個空行，發送回車符和換行符，通知服務器以下不再有響應頭部。

　　響應包體：服務器返回給客戶端的文本信息;

　　HTTP 工作原理

　　HTTP 協議採用請求/響應模型。客戶端向服務器發送一個請求報文，服務器以一個狀態作為響應。

　　以下是 HTTP 請求/響應的步驟：

　　● 客戶端連接到web服務器：HTTP 客戶端與web服務器建立一個 TCP 連接;

　　● 客戶端向服務器發起 HTTP 請求：通過已建立的TCP 連接，客戶端向服務器發送一個請求報文;

　　● 服務器接收 HTTP 請求並返回 HTTP 響應：服務器解析請求，定位請求資源，服務器將資源副本寫到 TCP 連接，由客戶端讀取;

　　● 釋放 TCP 連接：若connection 模式為close，則服務器主動關閉TCP 連接，客戶端被動關閉連接，釋放TCP 連接;若connection 模式為keepalive，則該連接會保持一段時間，在該時間內可以繼續接收請求;

　　● 客戶端瀏覽器解析HTML內容：客戶端將服務器響應的 html 文本解析並顯示;

　　例如：在瀏覽器地址欄鍵入URL，按下回車之後會經歷以下流程：

　　1、瀏覽器向[DNS 服務器](http://www.chinabyte.com/keyword/DNS%E6%9C%8D%E5%8A%A1%E5%99%A8/)請求解析該 URL 中的域名所對應的 IP 地址;

　　2、解析出 IP 地址後，根據該 IP 地址和默認端口 80，和服務器建立 TCP 連接;

　　3、瀏覽器發出讀取文件(URL 中域名後面部分對應的文件)的HTTP 請求，該請求報文作為 TCP 三次握手的第三個報文的數據發送給服務器;

　　4、服務器對瀏覽器請求作出響應，並把對應的 html 文本發送給瀏覽器;

　　5、釋放 TCP 連接;

　　6、瀏覽器將該 html 文本並顯示內容;

　　HTTP 無狀態性

　　HTTP 協議是無狀態的(stateless)。也就是說，同一個客戶端第二次訪問同一個服務器上的頁面時，服務器無法知道這個客戶端曾經訪問過，服務器也無法分辨不同的客戶端。HTTP 的無狀態特性簡化了服務器的設計，使服務器更容易支持大量並發的HTTP 請求。

　　HTTP 持久連接

　　HTTP1.0 使用的是非持久連接，主要缺點是客戶端必須為每一個待請求的對象建立並維護一個新的連接，即每請求一個文檔就要有兩倍RTT 的開銷。因為同一個頁面可能存在多個對象，所以非持久連接可能使一個頁面的下載變得十分緩慢，而且這種短連接增加了傳輸的負擔。HTTP1.1 使用持久連接keepalive，所謂持久連接，就是服務器在發送響應後仍然在一段時間內保持這條連接，允許在同一個連接中存在多次數據請求和響應，即在持久連接情況下，服務器在發送完響應後並不關閉TCP 連接，而客戶端可以通過這個連接繼續請求其他對象。

　　HTTP/1.1 協議的持久連接有兩種方式：

　　● 非流水線方式：客戶在收到前一個響應後才能發出下一個請求;

　　● 流水線方式：客戶在收到 HTTP 的響應報文之前就能接著發送新的請求報文;

最後給出一個具體例子：

　　Remote[Address](http://www.chinabyte.com/keyword/address/):116.57.254.104:80

Request URL:http://hr.tencent.com/ Request

Method:GET

Status Code:200 OK Request Headers GET / HTTP/1.1

Host: hr.tencent.com

Connection: keep-alive

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

User-Agent:[Mozilla](http://www.chinabyte.com/keyword/Mozilla/)/5.0 (X11;[Linux](http://www.chinabyte.com/keyword/Linux/) i686) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.114 Safari/537.36

Accept-Encoding: gzip,deflate,sdch

Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4

Cookie: pgv\_pvi=2098703360; PHPSESSID=bc7onl0dojbsatscsfv79pds77; pgv\_info=ssid=s1454606128; pgv\_pvid=926725350; ts\_uid=4084753309

Response Header HTTP/1.1 200 OK

Server: nginx Date: Mon, 26 Jan 2015 01:09:10 GMT

Content-Type: text/html;charset=utf-8

Content-Length: 3631

Connection: keep-alive

X-Powered-By: PHP/5.3.10

Expires: Thu, 19 Nov 1981 08:52:00 GMT

Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0

Pragma: no-cache

Vary: Accept-Encoding

Content-Encoding: gzip

　　從請求報文可以知道：

　　GET / HTTP/1.1

　　請求方法 GET 表示一個讀取請求，將從服務器獲得網頁數據，/表示URL 的路徑，URL 總是以/開頭，/就表示首頁，最後的HTTP/1.1 指示採用的 HTTP 協議版本是 1.1;請求域名如下所示：

　　Host: hr.tencent.com

　　響應報文如下：

　　HTTP/1.1 200 OK

1. Ajax get請求

*// 設置響應頭  這個\*是Access-Controll-Allow-Origin的value*

*res*.setHeader('Access-Control-Allow-Origin', '\*');

xhr.onreadystatechange 是一個狀態監聽器

xhr.readyState有 0 1 2 3 4 五種狀態

0 是onreadystatechange 一開始(未初始化)

1 是open() 完結

2是 send() 完結

3 是服務端返回部分結果

4 是服務端返回所有結果

xhr.status 響應狀態碼

http://是網頁伺服器與你的電腦瀏覽器，以一般(非安全)模式在進行互動交談，所以內容有可能遭攔截竊聽；換句話說，你在此類網頁上填寫傳送的資料有可能被有心人士看到。

https://多了一個字母S的差別代表"安全(secure)"，基本上意謂著，你的電腦與伺服器間的資料傳遞是以加密的方式進行進行互動交談。

獲取 **響應報文**內容

console.log(xhr.status); *// 狀態碼  200*

console.log(xhr.statusText); *// 狀態字符串  OK*

console.log(xhr.getAllResponseHeaders()); *// 所有響應頭*

console.log(xhr.response); *// 響應體*

post的參數放xhr.send(xxx)

get的參數放url內 http://google.com/login?username=100&password=12345

response.setHeader(‘頭名’,’value’) 是響應頭

xhr.setRequestHeader(‘頭名’,’value’) 是請求頭

頭名 eg: content-type,Access-Control-Allow-Origin

content-type 是設置 內容類型

傳get參數要用 content-type , application/x-www-form-urlencoded

xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

3-JSON.html

request send去服務端的多數都是JSON類型數據。

所以我們可以直接用JSON方法。

NodeJS server端的

response.send() 只能傳 String與buffer (二進制的東西)。

服務端

*res*.setHeader('Access-Control-Allow-Headers', '\*');

    var data = { name: "json-server's response!" };

    var str = JSON.stringify(data);

*res*.send(str);

客戶端(browser)

if (xhr.status >= 200 && xhr.status < 300) {

                        let data = JSON.parse(xhr.response);

                        result.innerHTML = data.name;

                        result2.innerHTML = xhr.response;

                    }

也可以自動轉換 responseType

 xhr.responseType = 'json';

就不用自己手動 data= JSON.parse(xhr.response);了

4-ie cache問題

xhr.open('GET', 'http://127.0.0.1:3000/ie?t=' + Date.now());

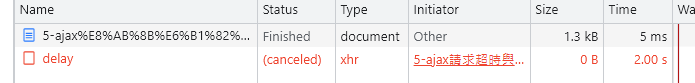
在請求路徑上加上get參數query ?t= 時間戮Date.now(); 這樣每次url都不同,ie就不會去跑緩存了。

5. 請求超時與網絡異常

*// 超時設置  2s*

xhr.timeout = 2000;

*// 意思是如果2秒之內沒有response,就取消request*



canceled 取消了request

*// 超時回調callback*

            xhr.ontimeout = function() {

                result.innerHTML = '網絡異常,2秒也沒有回應'

            };

xhr.timout = time; 因超時而取消請求

xhr.ontimout = function(){xxx}; 因超時而取消請求的callback

xhr.onerror = function(){xxx}; request-error時的callback

*// 網絡異常回調*

            xhr.onerror = function() {

                alert('something went WRONG! on error');

            }

xhr.onerror 會在客戶端自己的網絡出問題時觸發回調。

eg: offline 沒有網。

6-取消請求.html

xhr.abort() 手動取消請求

因為在不同作用域, xhr要先放到全局區。

let xhr = new XMLHttpRequest();

        btn.onclick = function() {

            xhr.open('GET', 'http://127.0.0.1:3000/delay');

            xhr.send();

            xhr.onreadystatechange = function() {

                if (xhr.readyState === 4) {

                    if (xhr.status >= 200 && xhr.status <= 299) {

                        result.innerHTML = xhr.response;

                    }

                }

            }

        }

        cancel.onclick = function() {

            xhr.abort();

        };

7-重複請求問題

用戶如果不停點同一個btn, 同一個請求多次重複發生,對服務器不好。

*// 防抖節流*

        let isSending = false;

btn.onclick = function() {

            if (isSending) {

                x.abort();

            };

xhr.onreadystatechange = function() {

                if (xhr.readyState === 4) {

*//  防抖節流*

                    isSending = true;

jQuery ajax p21 p22 需要再看

<https://www.bilibili.com/video/BV1WC4y1b78y?p=23>

axios github api

<https://github.com/axios/axios>

引入axios in html

<script src="../node\_modules/axios/dist/axios.min.js"></script>

不要加type=”module” 會報錯

*// axios配置 baseURL*

        axios.defaults.baseURL = 'http://127.0.0.1:3000'

axios.get(url,[{params,headers,data}])

btns[0].onclick = function() {

            axios.get('/axios-server', {

*// url 參數 params*

                params: {

                    id: 1,

                    price: 100

                }

*// 請求頭 headers*

*// headers: {*

*//     name: 'peter',*

*//     ages: '24'*

*// }*

            });

        };

axios 方法

axios.request(config)

axios.get(url[, config])

axios.delete(url[, config])

axios.head(url[, config])

axios.options(url[, config])

axios.post(url[, data[, config]])

axios.put(url[, data[, config]])

axios.patch(url[, data[, config]])

axios.getUri([config])

axios中

params 是url.query , get的url參數

data 是post的body , post請求體

headers是請求頭 setRequestHeader

axios的response

.then(function(*response*) {

// axios的response有全部內容 我們取響應體 data

                let res = JSON.stringify(*response*.data);

                result.innerHTML = res;

            })

會返回全部 response內容, status,statusText, headers, data

其中data 是響應體。 response body

axios() 寫法 對象寫法,ajax也有

axios({

url: '/getUsers',

method: 'get',

responseType: 'json', // default

data: {

//'a': 1,

//'b': 2,

}

}).then(function (response) {

console.log(response);

console.log(response.data);

}).catch(function (error) {

console.log(error);

}）

9-fetch發送ajax請求

fetch是一個全局對象(直接調用就可,不用引入) 返回值是promise

Promise<Response> fetch(input[, init]);

### [參數](https://developer.mozilla.org/zh-CN/docs/Web/API/fetch#%E5%8F%82%E6%95%B0)

**?input**

定義要獲取的資源。這可能是：

* 一個 [USVString](https://developer.mozilla.org/zh-CN/docs/Web/API/USVString) 字符串，包含要獲取資源的 URL。一些瀏覽器會接受 blob: 和 data: 作為 schemes.
* 一個 [Request](https://developer.mozilla.org/zh-CN/docs/Web/API/Request) 對象。

**init 可選**

一個配置項對象，包括所有對請求的設置。可選的參數有：

* method: 請求使用的方法，如 GET、POST。
* headers: 請求的頭信息，形式為 [Headers](https://developer.mozilla.org/zh-CN/docs/Web/API/Headers) 的對象或包含 [ByteString](https://developer.mozilla.org/zh-CN/docs/conflicting/Web/JavaScript/Reference/Global_Objects/String) 值的對象字面量。
* body: 請求的 body 信息：可能是一個 [Blob](https://developer.mozilla.org/zh-CN/docs/Web/API/Blob)、[BufferSource (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/BufferSource" \o "Currently only available in English (US))、[FormData](https://developer.mozilla.org/zh-CN/docs/Web/API/FormData)、[URLSearchParams](https://developer.mozilla.org/zh-CN/docs/Web/API/URLSearchParams) 或者 [USVString](https://developer.mozilla.org/zh-CN/docs/Web/API/USVString) 對象。注意 GET 或 HEAD 方法的請求不能包含 body 信息。
* mode: 請求的模式，如 cors、 no-cors 或者 same-origin。
* credentials: 請求的 credentials，如 omit、same-origin 或者 include。為了在當前域名內自動發送 cookie ， 必須提供這個選項， 從 Chrome 50 開始， 這個屬性也可以接受 [FederatedCredential (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/FederatedCredential" \o "Currently only available in English (US)) 實例或是一個 [PasswordCredential (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/PasswordCredential" \o "Currently only available in English (US)) 實例。
* cache:  請求的 cache 模式: default、 no-store、 reload 、 no-cache 、 force-cache 或者 only-if-cached 。
* redirect: 可用的 redirect 模式: follow (自動重定向), error (如果產生重定向將自動終止並且拋出一個錯誤）, 或者 manual (手動處理重定向). 在Chrome中默認使用follow（Chrome 47之前的默認值是manual）。
* referrer: 一個 [USVString](https://developer.mozilla.org/zh-CN/docs/Web/API/USVString) 可以是 no-referrer、client或一個 URL。默認是 client。
* referrerPolicy: 指定了HTTP頭部referer字段的值。可能為以下值之一： no-referrer、 no-referrer-when-downgrade、 origin、 origin-when-cross-origin、 unsafe-url 。
* integrity: 包括請求的  [subresource integrity](https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity) 值 （ 例如： sha256-BpfBw7ivV8q2jLiT13fxDYAe2tJllusRSZ273h2nFSE=）。

### [返回值](https://developer.mozilla.org/zh-CN/docs/Web/API/fetch#%E8%BF%94%E5%9B%9E%E5%80%BC)

一個 [Promise](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise)，resolve 時回傳 [Response](https://developer.mozilla.org/zh-CN/docs/Web/API/Response) 對象。

fetch的config

headers,

method,

body(axois data)

\*\*fetch沒有params(url query) 需要自己在url上寫入。

preflight

一般來說 preflight 預檢請求，指的就是 OPTIONS 請求。 它會在瀏覽器認為 即將要執行的請求可能會對服務器造成不可預知的影響時 ，由瀏覽器自動發出。 通過預檢請求，瀏覽器能夠知道當前的服務器是否允許執行即將要進行的請求，只有獲得了允許，瀏覽器才會真正執行接下來的請求。

\*\*\*

fetch返回值promise內value (response)內的[[prototype]]有一個text()方法,用來獲取response body的 響應體內容。

用法:

btn.onclick = function() {

      fetch('http://127.0.0.1:3000/fetch-server?vip=10&price=2000', {

*// config*

                method: 'post',

                headers: {

                    name: 'fetch-peter'

                },

                body: {

                    d1: 'bodyData!'

                }

            }).then(*response* => {

                return *response*.text();

            }).then(*response* => {

                result.innerHTML = *response*;

            })

        }

\*\*\*\*\*

Request Payload更準確的說是http request的payload body。一般用在數據通過POST請求或者PUT請求。它是HTTP請求中*空行*的後面那部分。（PS:這裡涉及一個http常被問到的問題，http請求由哪幾部分組成，一般是請求行，請求頭，空行，請求體。payload body應該是對應請求體。）

一個請求伴隨著header設置為Content-Type: application/json時候，看起來可能像這樣：

POST /some-path HTTP/1.1

Content-Type: application/json

{ "foo" : "bar", "name" : "John" }

如果你正常請求一個ajax。瀏覽器會簡單的將你提交的內容作為payload展示出來，這就是它所能做的，因為它不知道數據來自哪裡。

如果你提交了一個html表單並且配置上了method="post"，並且設置了Content-Type: application/x-www-form-urlencoded或者Content-Type: multipart/form-data。那麼你的請求可能長這個樣：

POST /some-path HTTP/1.1

Content-Type: application/x-www-form-urlencoded

foo=bar&name=John

這裡的form-data就是request payload。在這裡，瀏覽器知道更多：它知道bar是提交表單的輸入字段foo的值。這就是它向你展示的。

所以區別就是，他們只是因為Content-Type設置的不同，並不是數據提交方式的不同，這兩種提交都會將數據放在message-body中。但是chrome瀏覽器的開發者工具會根據這個ContentType區分顯示方式。

\*\*\*\*\*

fetch的body(post form-data) 如果以Object型式傳到服務端會變成數組

[obj obj]

而且無論如何傳, fetch的body (form-data)到了服務端都變成了key,沒有value。



即使傳對象, 到了服務端也變成了 空value。

## 同源政策(Same Origin Policy)

是瀏覽器的一種安全策略。

同源: 協議,域名,端口號port 必須完全相同

違背同源就是跨域。

http向https請求 也是跨域。 (不同協議)

express

res.sendFile(path [, options] [, fn])

Node.js 中，\_\_dirname 總是指向被執行 js 文件的絕對路徑，所以當你在 /d1/d2/myscript.js 文件中寫了 \_\_dirname， 它的值就是 /d1/d2 。

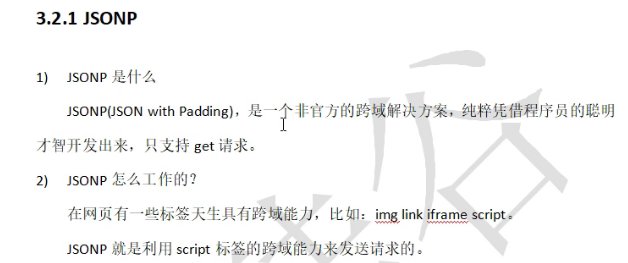
相反，./ 會返回你執行 node 命令的路徑，例如你的工作路徑。

有一個特殊情況是在 require() 中使用 ./ 時，這時的路徑就會是含有 require() 的腳本文件的相對路徑。

*res*.sendFile(\_\_dirname + '/index.html');

3.2 **如何解決跨域**

JSONP(JSON with Padding) 是一個非官方的跨域解決方案, 純粹由程序員開發出來, 只支持get請求。



比如: 跨域方法 引入axios

<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

這就是跨域。 利用script標簽。

response 傳回客戶端的一定要是 js格式的字符串,

這樣瀏覽器,<script>標簽才能夠執行。 比如:

*res*.send('console.log("hello by jsonp")');

服務端

*// jsonp-server*

app.all('/jsonp-server', function(*req*, *res*) {

*// res.send('console.log("hello by jsonp")');*

    const data = {

        name: 'peter from jsonp'

    }

    let strData = JSON.stringify(data);

*// res.end(`handle(${strData})`);*

*res*.end('handle(' + strData + ')');

});

客戶端

<div id="result"></div>

*<!-- // 利用script標簽做請求 -->*

    <script>

        function handle(*data*) {

            const result = document.querySelector('#result');

            result.innerHTML = *data*.name;

        }

    </script>

    <script src="http://127.0.0.1:3000/jsonp-server">

    </script>

*<!-- 這個引入script標簽以 get請求,*

*獲取了handle('peter from jsonp')  字符串*

*給js瀏覽器讀取 -->*

2.3 總結原生JS實現JSONP的步驟\*\*\*\*\*

2.3.1 客戶端

1.定義獲取數據後調用的回調函數

2.動態生成對服務端JS進行引用的代碼

2.0.1創建script標簽，並設置其src屬性

2.0.2設置url為提供jsonp服務的url地址，並在該url中設置相關callback參數

2.0.3把script標簽加入document 文檔，此時調用開始。

2.3.2 服務端

將客戶端發送的callback參數作為函數名來包裹住JSON數據，返回數據至客戶端。

官方跨域解決方案

跨來源資源共用（Cross-Origin Resource Sharing ([CORS](https://developer.mozilla.org/zh-TW/docs/Glossary/CORS))）是一種使用額外 [HTTP](https://developer.mozilla.org/zh-TW/docs/Glossary/HTTP) 標頭令目前瀏覽網站的[使用者代理 (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/User_agent)取得存取其他來源（網域）伺服器特定資源權限的機制。當使用者代理請求一個不是目前文件來源——例如來自於不同網域（domain）、通訊協定（protocol）或通訊埠（port）的資源時，會建立一個**跨來源 HTTP 請求（cross-origin HTTP request）**。



*res*.setHeader('Access-Control-Allow-Headers', '\*');

\*是全部。

也可以指定 ‘http://127.0.0.1:3000’ 指定這個url才可以CORS。

<https://developer.mozilla.org/zh-TW/docs/Web/HTTP/CORS>

## [HTTP 回應標頭](https://developer.mozilla.org/zh-TW/docs/Web/HTTP/CORS#http_%E5%9B%9E%E6%87%89%E6%A8%99%E9%A0%AD)

這個小節列出了伺服器回傳予取存控制請求之由跨來源資源共用規範所定義的 HTTP 回應標頭。上一節已提供了這些行為的概述。

eg:

[Access-Control-Expose-Headers (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Expose-Headers)標頭表示伺服器允許瀏覽器存取回應標頭的白名單，如：

Access-Control-Expose-Headers: X-My-Custom-Header, X-Another-Custom-Header

Copy to Clipboard

這允許了瀏覽器能夠存取回應當中的 X-My-Custom-Header 以及 X-Another-Custom-Header 標頭。

### [Access-Control-Allow-Origin](https://developer.mozilla.org/zh-TW/docs/Web/HTTP/CORS#access-control-allow-origin)

一個回應的資源可能擁有一個 [Access-Control-Allow-Origin (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin) 標頭，如以下的語法：

Access-Control-Allow-Origin: <origin> | \*

Copy to Clipboard

origin 參數指定了一個可以存取資源的 URI。瀏覽器必定會執行此檢查。對一個**不帶有**身分驗證的請求，伺服器可以指定一個「\*」作為萬用字元（wildcard），從而允許任何來源存取資源。

### [Access-Control-Allow-Methods](https://developer.mozilla.org/zh-TW/docs/Web/HTTP/CORS#access-control-allow-methods)

[Access-Control-Allow-Methods (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Methods) 標頭表示存取資源所允許的方法，用來回應預檢請求。上面已討論請求之預檢的條件。

Access-Control-Allow-Methods: <method>[, <method>]\*

默認允許get, post。

如果我們需要用到delete,可能就要設置一下access-control-allow-methods。

Access-Control-Allow-Origin: http://foo.example

Access-Control-Allow-Methods: POST, GET, OPTIONS

Access-Control-Allow-Headers: X-PINGOTHER, Content-Type

Access-Control-Max-Age: 86400

Copy to Clipboard

伺服器回應中的 Access-Control-Allow-Methods 標頭表示伺服器可以接受 POST、GET 和 OPTIONS 方法。

### [Access-Control-Allow-Headers](https://developer.mozilla.org/zh-TW/docs/Web/HTTP/CORS#access-control-allow-headers)

[Access-Control-Allow-Headers (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Headers) 標頭用在回傳予預檢請求的回應當中，以指定哪些 HTTP 標頭可以於實際請求中使用。

星號\* 可以傳任何自定義的 請求頭。

常用的有:

*res*.setHeader('Access-Control-Allow-Origin', '\*');

*res*.setHeader('Access-Control-Allow-Headers', '\*');

*res*.setHeader('Access-Control-Allow-Methods','\*');