



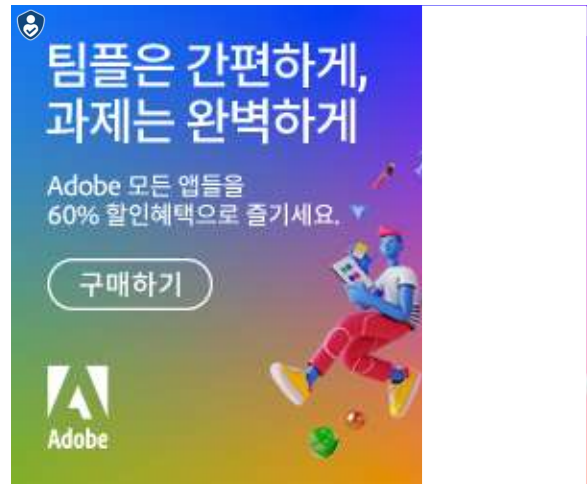
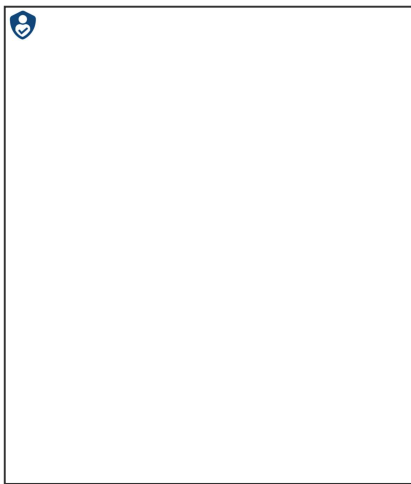
백엔드 과정, 포기하지 않고
5개월 간 제대로 배워보실 분 모집!



강좌2 - NodeJS - 6년 전 등록 / 3년 전 수정

서버 만들기

Node.js 교과서 책이 출간되었습니다. 이 포스팅보다는 책이나 동영상 강의를 보시는 것을 추천합니다.



안녕하세요! 이번 시간에는 **서버**를 만들어보겠습니다!

서버를 직접 만든다니 새롭지 않나요? 다른 서버들은 서버를 다운로드받은 후 설정... 설정... 설정... 끝해야 돌릴 수 있잖아요. **Node.js**는 코드를 통해 서버를 만들고 실행할 수 있습니다. (물론 설정하는 것은 자신의 몫입니다.) 웹서버가 안에 내장되어 있기 때문에 가능합니다.

서버

우선 `server.js` 파일을 만들고, 그 안에 다음과 같이 써줍니다.

`server.js`

```
1 const http = require('http'); // 서버를 만드는 모듈 불러옴
2 http.createServer((request, response) => { // 서버 만드는 메소드
3   console.log('server start!');
4 }).listen(8080);
```

잠깐 코드를 볼까요? Node.js는 **모듈** 시스템을 구축하고 있습니다. 모듈이란, 필요한 것만 불러오는 것이라고 생각하면 됩니다. http 서버가 필요하니 **http** 모듈을 **require** 메소드를 통해서 불러와 http 변수에 저장하였습니다. 전 시간



목록





어주면 됩니다. 그리고 8080번 포트에 연결(**listen**)합니다.

그런 후 **cmd**를 실행하고 *server.js*를 만든 곳으로 이동합니다. 이동하는 방법은 **cd**라는 명령어로 합니다.

```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd ..

C:\Windows>cd ..

C:\>cd users

C:\Users>cd zerocho

C:\Users\ZeroCho>
```

위의 사진을 보세요. **cd [폴더이름]** 으로 원하는 폴더로 이동할 수 있습니다. 상위 폴더로 가는 것은 **cd ..**하면 됩니다. 사진에서는 상위 폴더로 두 번 이동하고 users폴더, zerocho폴더 순으로 들어갔습니다. 이렇게 각자 *server.js*를 만든 폴더로 들어가면 됩니다.

그 폴더까지 갔다면 **node server.js**를 쳐보세요. *server.js* 파일을 구동하는 명령어입니다. 그리고 웹 브라우저로 <http://localhost:8080>에 들어가면, cmd에 'server start!' 라고 아래와 같이 뜹니다. 콘솔 명령어만 사용했고, 아직 브라우저에는 아무것도 보내지 않았기 때문에 브라우저에는 아무것도 뜨지 않습니다.

```
C:\Users\ZeroCho>node server.js
server start!
```

만약 8080번 포트 대신에 그냥 일반 주소로 하고 싶다면, **80**번 포트에 listen하게 하면 됩니다. 80번 포트는 기본이라 :80을 쓰지 않아도 됩니다. 네이버같은 사이트도 다 80번 포트를 이용하고 있기 때문에 www.naver.com:8080 이렇게 입력하지 않아도 되는 겁니다. 대신 80포트는 이미 다른 프로그램에서 사용하고 있는 경우가 많아 충돌이 자주 발생합니다. 자기 컴퓨터로 할 때는 80포트 대신 다른 포트를 쓰고, 서버에 올릴 때는 80포트로 올리는 게 좋습니다.

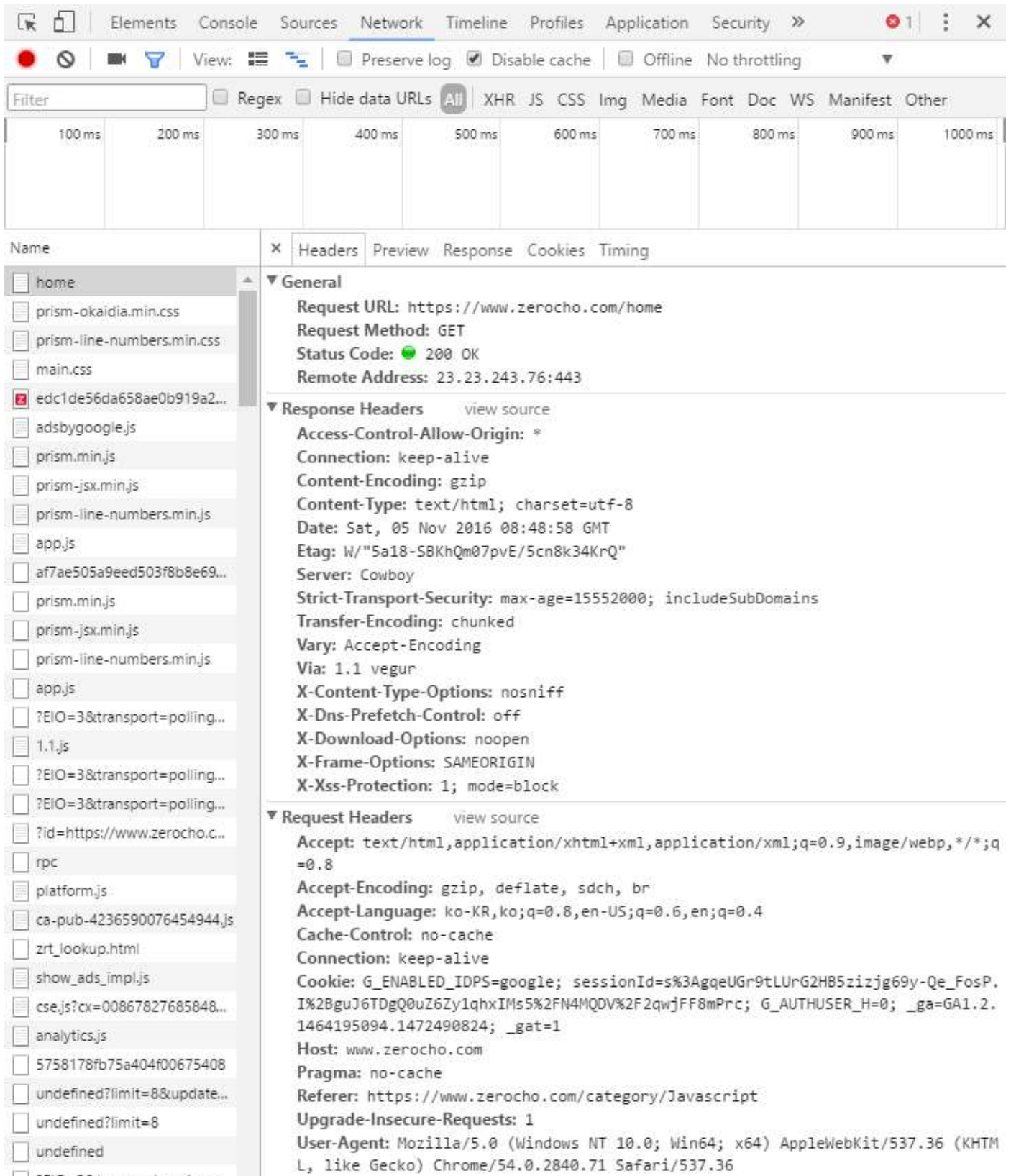
Request, Response

이제 `createServer` 메소드 콜백의 매개변수인 **request**와 **response**에 대해서 알아보죠. **request**는 요청을 담당합니다. 즉 서버로 보내는 요청에 대한 정보가 들어있습니다. 방금 우리가 주소창에 localhost:8080이라고 친 행위도 서버에 그 주소에 해당하는 **정보**를 달라고 요청한 겁니다. 서버는 항상 대기중이다가 request가 들어올 때 반응하면 됩니다. 그리고 request에 대한 처리를 한 후 결과를 **response** 객체로 돌려주는 겁니다.

response는 이제 클라이언트(브라우저)로 돌려줄 응답을 담당합니다. 만약 어떤 정보를 보내고 싶다면 response 객체를 활용하면 됩니다. 지금까지 브라우저에 아무것도 안 뜬 이유는 response를 설정하지 않았기 때문입니다. response



는 진짜 주고받고자 하는 내용이 들어있습니다. 제 홈페이지 주소를 주소창에 입력했을 때의 request와 response의 header 내용입니다. F12(개발자도구)를 누르고 Network 탭에 가면 볼 수 있습니다.



The screenshot shows the Chrome DevTools Network tab. The left pane lists various resources, with 'home' selected. The right pane shows the 'Headers' tab for the selected resource. The 'General' section displays the request URL, method, status code (200 OK), and remote address. The 'Response Headers' section lists various headers including Access-Control-Allow-Origin, Connection, Content-Encoding, Content-Type, Date, Etag, Server, Strict-Transport-Security, Transfer-Encoding, Vary, Via, X-Content-Type-Options, X-Dns-Prefetch-Control, X-Download-Options, X-Frame-Options, and X-Xss-Protection. The 'Request Headers' section lists headers such as Accept, Accept-Encoding, Accept-Language, Cache-Control, Connection, Cookie, Host, Pragma, Referer, Upgrade-Insecure-Requests, and User-Agent.

General

- Request URL: `https://www.zerocho.com/home`
- Request Method: `GET`
- Status Code: `200 OK`
- Remote Address: `23.23.243.76:443`

Response Headers

- Access-Control-Allow-Origin: `*`
- Connection: `keep-alive`
- Content-Encoding: `gzip`
- Content-Type: `text/html; charset=utf-8`
- Date: `Sat, 05 Nov 2016 08:48:58 GMT`
- Etag: `W/"5a18-58KhQm07pvE/5cn8k34KrQ"`
- Server: `Cowboy`
- Strict-Transport-Security: `max-age=15552000; includeSubDomains`
- Transfer-Encoding: `chunked`
- Vary: `Accept-Encoding`
- Via: `1.1 vegur`
- X-Content-Type-Options: `nosniff`
- X-Dns-Prefetch-Control: `off`
- X-Download-Options: `noopen`
- X-Frame-Options: `SAMEORIGIN`
- X-Xss-Protection: `1; mode=block`

Request Headers

- Accept: `text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8`
- Accept-Encoding: `gzip, deflate, sdch, br`
- Accept-Language: `ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4`
- Cache-Control: `no-cache`
- Connection: `keep-alive`
- Cookie: `G_ENABLED_IDPS=google; sessionId=s%3AgqeUGr9tLURG2HB5zizjg69y-Qe_FosP.I%2BguJ6TDgQ0uZ6Zy1qhxIMs5%2FN4MQDV%2F2qwJFF8mPrc; G_AUTHUSER_H=0; _ga=GA1.2.1464195094.1472490824; _gat=1`
- Host: `www.zerocho.com`
- Pragma: `no-cache`
- Referer: `https://www.zerocho.com/category/javascript`
- Upgrade-Insecure-Requests: `1`
- User-Agent: `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36`

온갖 정보들이 나오는 데 이것은 사용하기 나름입니다. 나중에 차차 알아가도록 하죠.

v7.3.0

```

4  .on('error', (err) => { // 요청에 에러가 있으면
5      console.error(err);
6  })
7  .on('data', (data) => { // 요청에 데이터가 있으면
8      console.log(data);
9  })
10 .on('end', () => { // 요청의 데이터가 모두 받아졌으면
11     response.on('error', (err) => { // 응답에 에러가 있으면
12         console.error(err);
13     });
14     response.statusCode = 200; // 성공 상태 코드
15     response.setHeader('Content-Type', 'text/plain'); // header 설정
16     response.write('hi\n'); // body에 정보 탑재
17     response.end('the end!'); // 정보 탑재 후 브라우저로 전송
18 });
19 }).listen(8080);

```

createServer 메소드의 인자인 response와 request를 사용하기 시작했습니다. 순서대로 살펴보면, 맨 처음은

`request.on('error', 콜백)` 입니다. 일단 요청에 **에러**가 있을 수도 있으니 처리하고 보는거죠. 에러가 생기면 서버가 멈춰버리기(죽어버린다고 표현합니다) 때문에 에러는 반드시 처리해줘야합니다.

사소한 에러 하나가 발생해서 서버가 죽었는데 그 사실을 모르고 있다면, 사용자들은 여러분의 서비스를 사용하지 못합니다. 그리고 대체 서비스를 찾겠죠. 이렇게 멀어진 관심은 다시 되돌리기 힘듭니다. ~~(제가 예전에 이 홈페이지에 에러가 난 것을 모르고 자버렸습니다)~~

다음은 `request.on('data', 콜백)`, `request.on('end', 콜백)`; 인데요. 각각 request에 data가 있을 경우 처리하는 부분과, data 처리가 다 끝났음을 알려주는 부분입니다. request에 data가 있는 경우는 HTML 폼 전송같이 폼의 내용을 전송하는 경우 데이터가 request의 data에 실려 서버로 전송됩니다.

요청이 끝났으니까 **응답**을 해줘야겠죠? end의 콜백에 response해줍니다. 일단 response에도 에러가 생길 수 있으니, `response.on('error', 콜백)`로 에러를 처리해줍니다. `response.statusCode`는 여러 개가 있는데 지금은 성공적으로 데이터를 전송했다는 것을 알려야하니까 **성공**의 의미인 **200**을 설정합니다. **statusCode에 대한 강좌** 그리고 response의 정보를 가진 header를 설정하는 **setHeader** 메소드를 사용해서 지금 보내는 콘텐츠의 유형을 평문 텍스트(text/plain)로 설정합니다. 만약 html을 전송하고 싶다면 text/html하면 됩니다. html 전송은 다음 시간에 알아보니다.

이제 response의 **body**를 설정해 정보를 보내주어야겠죠? 위의 예시에는 두 메소드가 있는데요. **write**와 **end**입니다. end 메소드는 write함과 동시에 response를 종료하는 겁니다. 종료하는 순간 클라이언트로 전송됩니다. write와 end 메소드로 hi\n, the end! 문자열을 전송했습니다. 이제 브라우저에는

```

hi
the end!

```

라는 짧은 결과가 표시됩니다. 아직은 서버가 좀 허접하죠? 다음 시간에는 html 파일을 만들어서 통째로 전송해봅시다!

이전글: [Node.js와 npm](#)

목록





투표로 게시글에 관해 피드백을 해주시면 게시글 수정 시 반영됩니다. 오류가 있다면 어떤 부분에 오류가 있는지도 알려주세요!
잘못된 정보가 퍼져나가지 않도록 도와주세요.

만족해요	24(80.00%)
설명이 부족해요	3(10.00%)
너무 어려워요	3(10.00%)
오류가 있는 거 같아요	0(0.00%)
총 투표 수	30

Copyright 2016- ZeroCho. 무단 전재 및 재배포 금지. 출처 표기 시 인용 가능.

이메일

비밀번호(수정 및 삭제 시 사용)

댓글을 입력하세요. 근거없는 비난 대신 날카로운 피드백을 원합니다. 답글이 달리면 삭제할 수 없습니다.

등록

3개의 댓글이 있습니다.

- 익명** 모듈이란 패키지와 같은 의미인가요? ↩ 2년 전
- ZeroCho** 보통 저는 패키지는 npm에서 다운 받는 것, 모듈은 require해오는 것들이라고 설명합니다. 모듈이 좀 더 넓은 범위네요. ↩ 2년 전
- 익명** 다소 멍청한 질문일수 있는데 궁금하게 생겼습니다. 본문 설명과 같이 nodejs와 npm등 환경을 구축하는 물리적인 서버가 필요할것 같은데 왜 serverless라고 칭하는지 궁금합니다. ↩ 2년 전
- ZeroCho** 저도 잘 모르겠는데 그냥 내가 운영하는 서버가 없다해서 서버리스인 것 같습니다. 실제로 서버는 어딘가에는 있죠. ↩ 2년 전
- 가즈아** 만족해요 누를려고 했는데 실수로 설명이 부족해요를 눌러버렸네요 .. ↩ 4년 전



목록



다양한 서비스를 직접 만드는, 실무에 가장 가까운 강좌

페이지 좋아요

메시지 보내기

타임라인

메시지


ZeroCho Blog
 약 한 달 전
<https://www.zerocho.com/cat.../post/61d064b6a7f9490004dd9358>

 ZEROCHO.COM
(etc) 2021년 ...
 [2020년 블로그 ...]

2개

댓글 달기

1개


ZeroCho Blog
 약 7개월 전
