

# JavaScript : 주차

비동기



# 목차

—  
프로미스에 대해 이해  
할 수 있다.

Async / await 를 사  
용할 수 있다

웹의 정보 통신에 대해  
이해할 수 있다.

01\_Promise

02\_Async / await

03\_웹 기초

04\_ajax



01

# Promise



## 👉 Callback

함수 A 호출에서 함수 B가 인자로 전달될 때 => 인자 값인 함수 B : 콜백 함수  
함수 A의 매개변수로 전달되어 A안에서 호출됨 : B의 실행 제어권을 A에게 전달

콜백 함수는 은 동기적으로, 비동기로 유용하게 쓰임

```
function func(cb) {  
  cb();  
}  
func(() => console.log('즉시 호출'));  
  
function asyncFunc(cb, second) {  
  setTimeout(cb, second * 1000);  
}  
asyncFunc(() => console.log('3초 뒤 실행'), 3);
```

## 👉 콜백함수의 단점 : 콜백 지옥

콜백 함수 안에 또 다른 콜백 함수로 연쇄적으로 연결되어 있는 형태

어디서 어떤 식으로 함수들이 연결 되어있는지 한눈에 파악 쉽지 않음

비즈니스 로직 직관적 이해 어려움

에러 , 디버깅, 문제 분석등의 유지보수가 어렵다 => 가독성이 떨어짐

```
function orderCoffee(phoneNumber, callback) {  
  getId(phoneNumber,function(id) {  
    getEmail(id,function(email) {  
      getName(email,function(name) {  
        order(name, 'coffee',function(result) {  
          callback(result);  
        });  
      });  
    });  
  });  
}
```

## 👉 콜백함수의 단점 : 비동기 함수 에러처리

에러가 전파되다 보니 비동기 함수에서 발생하는 에러를 처리하지 못함

```
112 try {  
113     setTimeout(() => {  
114         throw new Error('에러 발생');  
115     }, 1000);  
116 } catch (error) {  
117     console.error('내가 에러를 잡을 꺼야!');  
118 }
```

문제 1 터미널 디버그 콘솔 출력

[Running] node "c:\javascript\_work\04week2\02.callback2.js"

c:\javascript\_work\04week2\02.callback2.js:114

```
    throw new Error('에러 발생');  
    ^
```

Error: 에러 발생

```
    at Timeout._onTimeout (c:\javascript_work\04week2\02.callback2.js:114:9)  
    at listOnTimeout (node:internal/timers:559:17)  
    at processTimers (node:internal/timers:502:7)
```



✔ Promise

콜백을 좀 더 깔끔하게 사용할 수 있다  
어떤 이벤트가 성공적으로 끝났는지 혹은 실패 했는지를 비동기로  
수행하고 그것의 결과값을 알려주는 객체

프로미스 상태 정보	의미	상태 변경 조건
pending	비동기 처리가 아직 수행되지 않은 상태	프로미스가 생성된 직후 기본 상태
fulfilled	비동기 처리가 수행된 상태( 성공 )	resolve 함수 호출
rejected	비동기 처리가 수행된 상태( 실패 )	reject 함수 호출

## ✔ Promise

```
49 console.log('start');  
50  
51 setTimeout(() => console.log('setTimeout'), 0);  
52  
53 Promise.resolve('Promise').then((res) => console.log(res));  
54  
55 console.log('End');  
56
```

문제 1 터미널 디버그 콘솔 출력 Code ▾

```
[Running] node "d:\projects\z06강예제\01.promise1.js"  
start  
End  
Promise  
setTimeout  
  
[Done] exited with code=0 in 0.176 seconds
```



## ④ Promise 객체

promise 생성자 함수는 빌트인 객체로 보통 비동기 처리를 수행할 콜백 함수를 인자값

1. 기본적으로는 pending 상태
2. 콜백 함수가 비동기 처리 호출
3. 비동기 처리 결과에 따라 프로미스 상태가 변경

비동기 처리 성공 → resolve 함수를 호출 → 프로미스를 fulfilled 상태로 변경

비동기 처리 실패 → reject 함수를 호출 → 프로미스를 rejected 상태로 변경



## ✔ Promise 객체

프로미스 객체는 는 비동기 처리 상태와 비동기 처리 결과도 상태로 갖는다

rejected 또는 fulfilled 상태를 settled 상태

```
const fulfilled = new Promise((resolve, reject) => resolve(1));

const rejected = new Promise((resolve, reject) =>
  reject(new Error('failure Error'))
);

console.log(rejected);
console.log(fulfilled);
```

```
> fulfilled
< ▼ Promise {<fulfilled>: 1} ⓘ
  ► [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: 1
> rejected
< Promise {<rejected>: Error: failure Error
  at <anonymous>:5:9
  ▼ at new Promise (<anonymous>)
    at <anonymous>:4:18} ⓘ
  ► [[Prototype]]: Promise
    [[PromiseState]]: "rejected"
  ► [[PromiseResult]]: Error: failure Error at <a
```



## ④ Promise 후속 처리 메서드

프로미스의 비동기 처리 상태가 변경되면, 이에 따른 후속 처리를 해야 함.

프로미스가 fulfilled 상태가 되면 → 성공 결과를 가지고 무언가를 수행

프로미스가 rejected 상태가 되면 → 에러 결과를 가지고 에러 처리를 수행

1. 프로미스의 비동기 처리 상태 완료
2. promise Result 객체가 후속 처리 메서드에 인수전달
3. 그에 따른 콜백 함수가 선택적으로 호출된다.

프로미스는 후속 처리 메서드 **then, catch, finally** 를 제공



## ④ then

then : 두개의 콜백 함수를 인자값 으로 전달 받음

첫 번째 콜백 함수

프로미스가 fulfilled 상태(resolve 함수가 호출된 상태)가 되면 호출,  
콜백 함수는 프로미스의 비동기 처리 결과를 인수로 전달받음

두 번째 콜백 함수

프로미스가 rejected 상태(reject 함수가 호출된 상태)가 되면 호출,  
콜백 함수는 프로미스의 에러를 인수로 전달받음



## ✓ catch

한 개의 콜백 함수를 인수로 전달받는다.

프로미스가 rejected 상태(reject 함수가 호출된 상태)인 경우만 호출  
catch 메서드는 then(undefined, onRejected) 과 동일하게 동작

```
new Promise((_, reject) => reject(new Error('rejected'))).catch((e) =>
  console.log(e)
); // Error: rejected

new Promise((_, reject) => reject(new Error('rejected'))).then(undefined, (e) =>
  console.log(e)
); // Error: rejected
```



## 👉 finally

한 개의 콜백 함수를 인수로 전달받는다.

프로미스의 성공 또는 실패와 상관없이 무조건 한 번 호출된다.

프로미스의 상태와 상관없이 공통적으로 수행해야 할 처리 내용이 있을 때 유용

```
function runTimer(seconds) {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      if (!seconds || seconds < 0) {  
        reject(new Error('0 보다 큰 ms초 단위가 들어와야한다'))  
      } else {  
        resolve(`${seconds} 초가 지났습니다`);  
      }  
    }, seconds * 1000);  
  });  
}  
  
runTimer(-2)  
  .then(() => console.log('타이머 종료'))  
  .catch(console.error)  
  .finally(() => console.log('프로그램 종료'));
```



## ④ Promise.all

프로미스는 직렬적으로 한 프로미스가 끝난 다음에 다음 프로미스를  
Promise.all 은 여러 비동기 태스크를 동시에(병렬적으로) 실행하고,  
가장 마지막 태스크가 완료될 때 완료 상태의 프로미스를 반환

1초 동작하는 프로미스 3초 동작하는 프로미스 => 총 4초

Promise.all 은 동시에 실행하여 => 총 3초

단 병렬적으로 실행하기때문에 한 개라도 reject 나오면 전체 에러로 표시된다

Promise.race : 가장 빨리 끝나는 task 만 수행 나머진 무시

Promise.allSettled : 성공한 task 실패한 task 전부 배열로 리턴



02

# async / await



### ④ async / await

async/await 키워드를 사용하면 **비동기 코드를 마치 동기 코드처럼** 보이게 작성  
try/catch로 일관되게 예외 처리를 할 수 있는 장점

#### promise 단점

Promise를 사용하면 try/catch 대신에 catch() 메서드를 사용하여 예외 처리 별도로 해 줌  
동기 코드와 비동기 코드가 섞여 있을 경우 예외 처리 가독성 떨어짐  
병렬적으로 처리할 때 then 안에 then 이 있어서 가독성 떨어짐

#### async 사용법

function 앞에 async 키워드 붙이기

Promise를 리턴 하는 함수 호출 앞에는 await 키워드 붙이기

async 키워드가 붙어있는 함수를 호출하면 자동으로 Promise 객체를 생성하여 리턴 함

## ✓ async / await

```
1  const myPromise = () => {
2    return new Promise((resolve, reject) => {
3      resolve('One');
4    });
5  };
6
7  const myFunc = async () => {
8    console.log('Inner function');
9    console.log(await myPromise());
10   console.log('goOut function');
11  };
12
13  console.log('Before Function');
14  myFunc();
15  console.log('After Function');
16
17  // myFunc().then(() => {
18  //   console.log('goOut function');
19  // });
```

문제 1 터미널 디버그 콘솔 출력 Code

[Running] node "d:\projects\z06강 예제 \06.async.js"

Before Function  
Inner function  
After Function  
One  
goOut function



## 📌 json : JavaScript Object Notation

서버와 클라이언트(브라우저, 모바일) 간의 http 통신을 위한 오브젝트 형태의 텍스트 포맷

"데이터이름": 값    예 => "name": "식빵"

1. JSON 데이터들은 쉼표(,)로 나열
2. 객체(object)는 중괄호({})로 둘러싸아 표현
3. 배열(array)은 대괄호([])로 둘러싸아 표현

값으로 올 수 있는 데이터 타입

숫자(number) ,문자열(string), 불리언(boolean),객체(object), 배열(array), NULL

참고

[http://www.tcpschool.com/json/json\\_basic\\_structure](http://www.tcpschool.com/json/json_basic_structure)

## ✓ json : JavaScript Object Notation

텍스트로 이루어진 json 파일 => 데이터를 받아서 객체나 변수에 할당하기 위해 사용

JavaScript 에서 Object => json 변환 => 서버로 전송

```
const json = JSON.stringify(객체이름);
```

서버에서 받은 json 데이터 => Object 로 변환

```
const obj = JSON.parse(json이름);
```

객체 안에 있는 메서드는 변환되지 않음!



03

# 웹 기초



## ☑ 인터넷과 웹

인터넷 - 전세계에 연결 되어있는 광역 네트워크 서비스

웹 - 인터넷 서비스 중 하나로 http 포트라고 하는 통신수단으로 정보를 주고 받고 하는 것  
즉 웹은 인터넷의 하위 항목이라고 보면 된다

서버 : 사용자들에게 서비스를 제공하는 컴퓨터

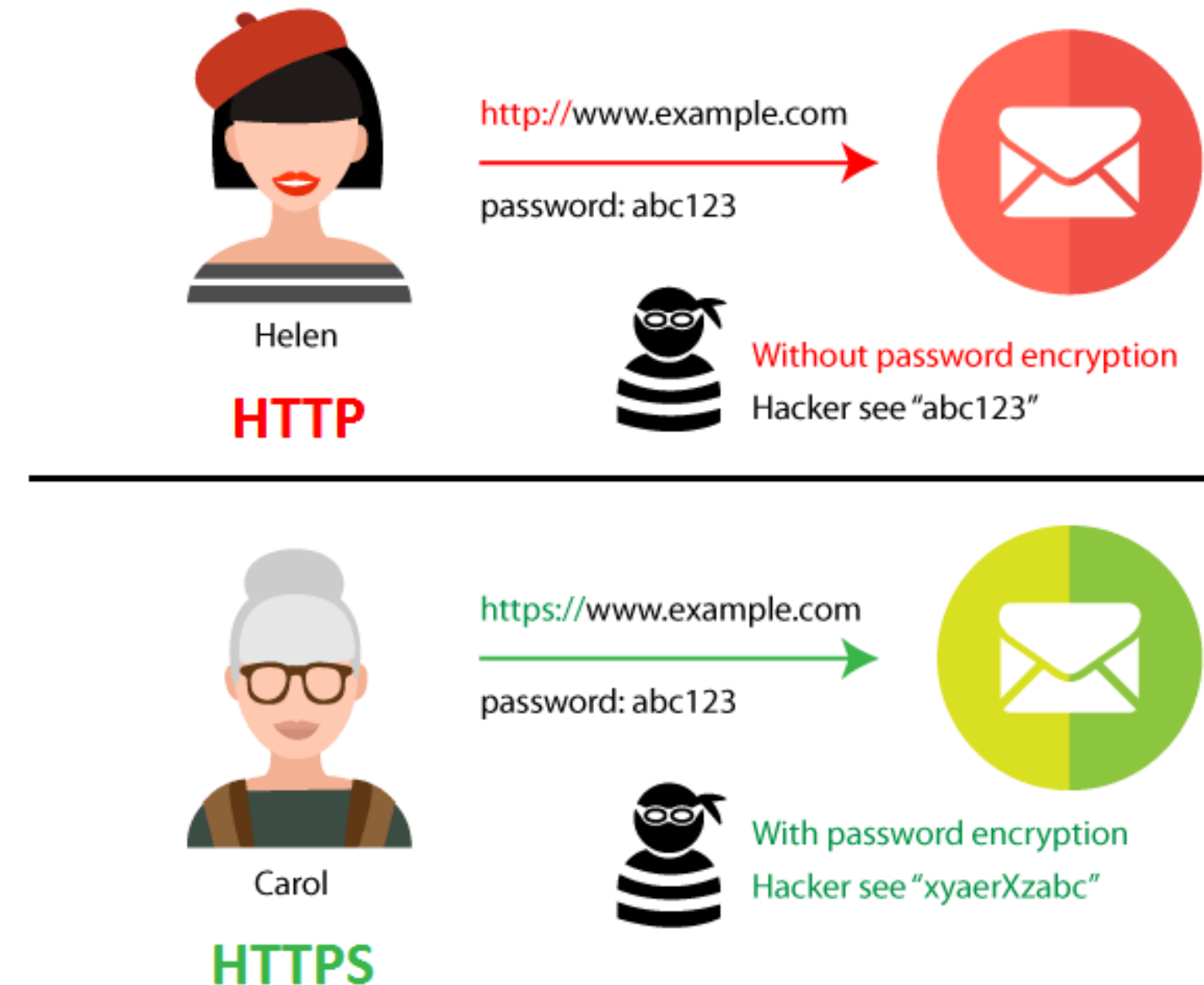
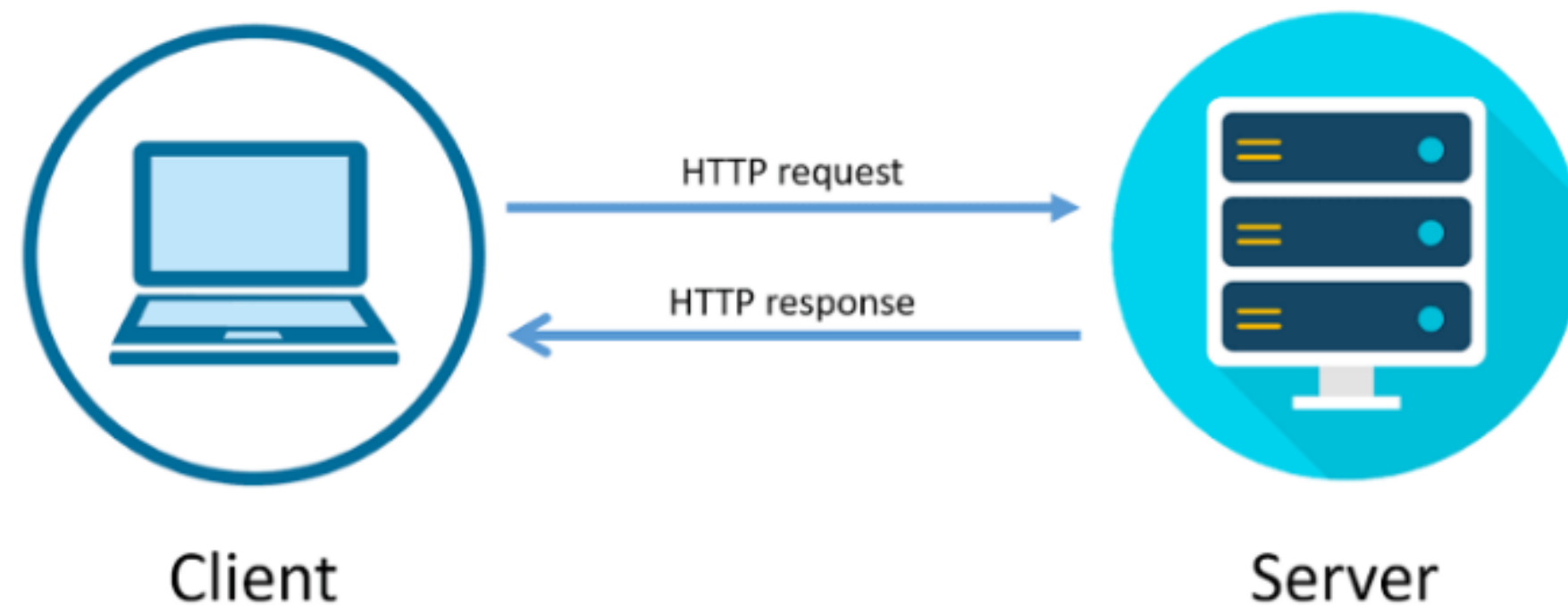
클라이언트 : 서버에게 서비스를 요청해서 사용하는 컴퓨터





## ☑ HTTP(hypertext transfer protocol)

HTML과 같은 하이퍼 미디어 문서를 전송하기 위한 request와 response 프로토콜  
브라우저와 웹 서버 간의 데이터 통신을 위해서 사용



참고

<https://developer.mozilla.org/ko/docs/Web/HTTP>

<https://seopressor.com/blog/http-vs-https/>

## ☑ IP 주소

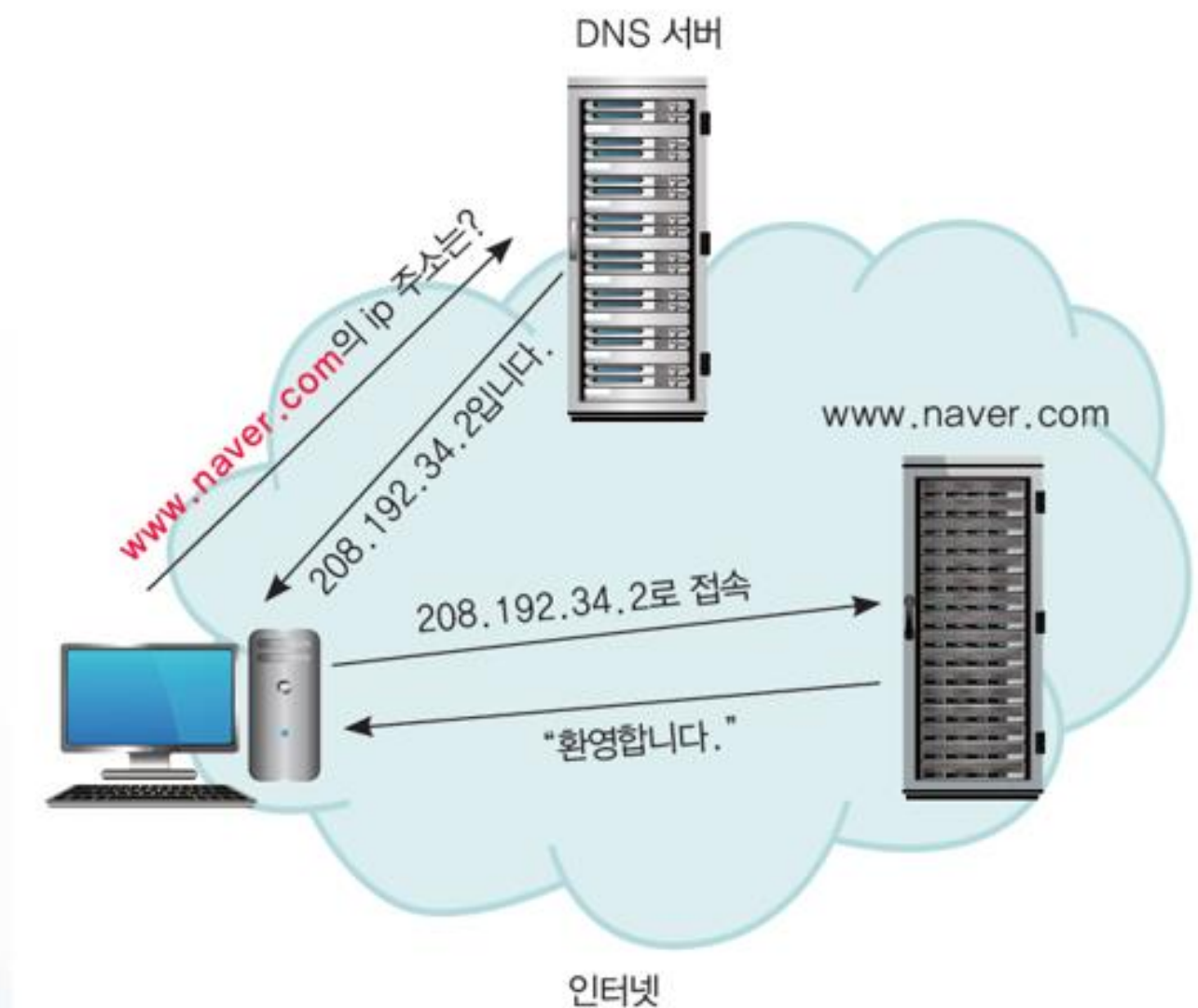
IP 주소 : 인터넷에서 컴퓨터의 주소

DNS(Domain Name System) : 숫자 대신 기호를 사용하는 주소

DNS 서버 : 기호 주소를 숫자 주소가 변환해주는 서버

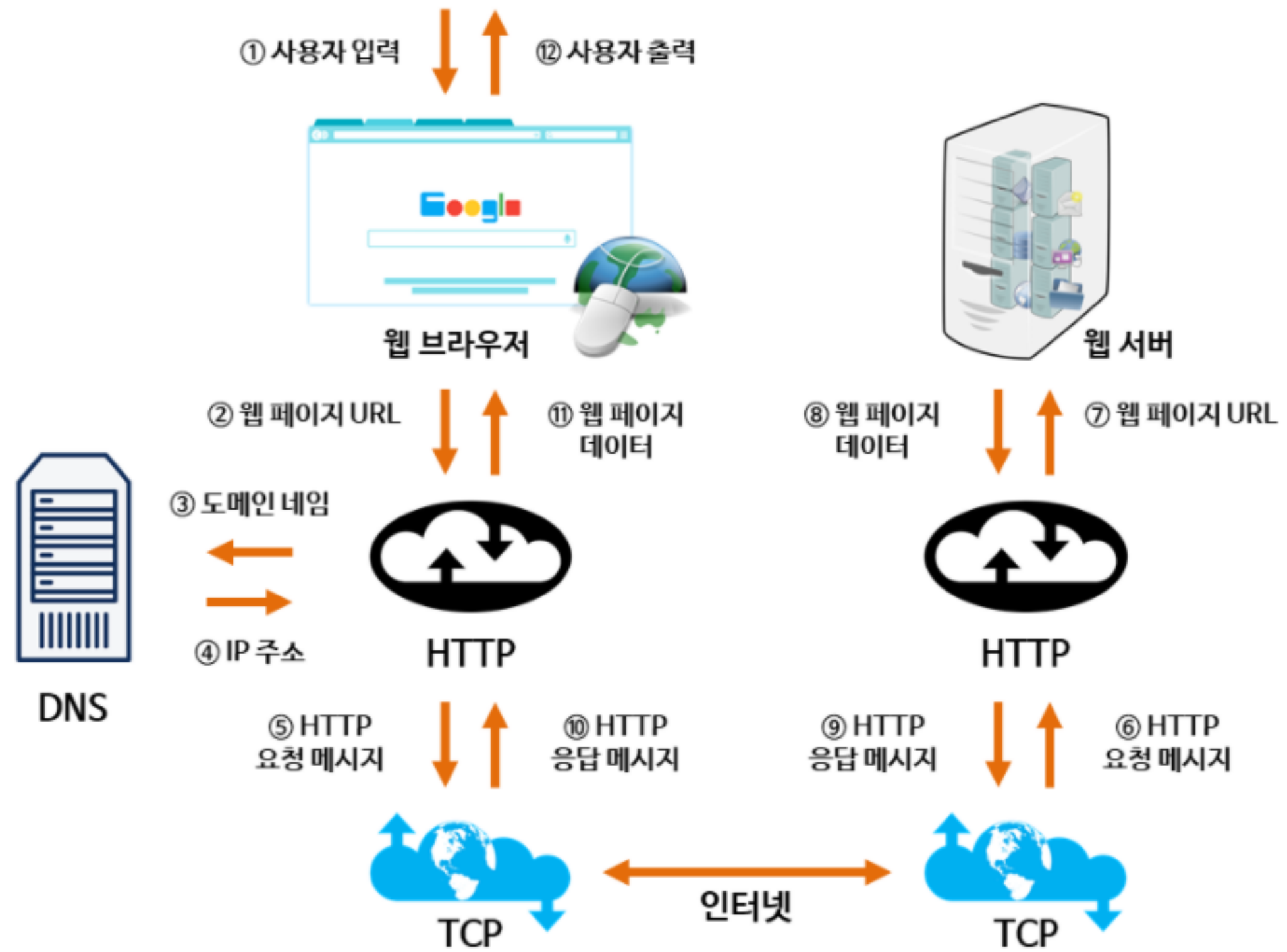
URL(Uniform Resource Locator)

인터넷 상의 파일이나 데이터베이스 같은 자원에 대한 주소 지정





## ④ 웹 동작 원리



## ☑ Status Codes

HTTP 응답 상태 코드는 특정 HTTP 요청이 성공적으로 완료 되었는지 실패했는지 상태에 대한 코드

총 5가지 범위가 있다

100 – 정보 제공

200 – 성공적인 처리

300 – 다른 곳으로 이동처리

400 – 클라이언트 잘못된 요청 처리

500 – 서버의 잘못된 응답 처리

## HTTP Status Codes

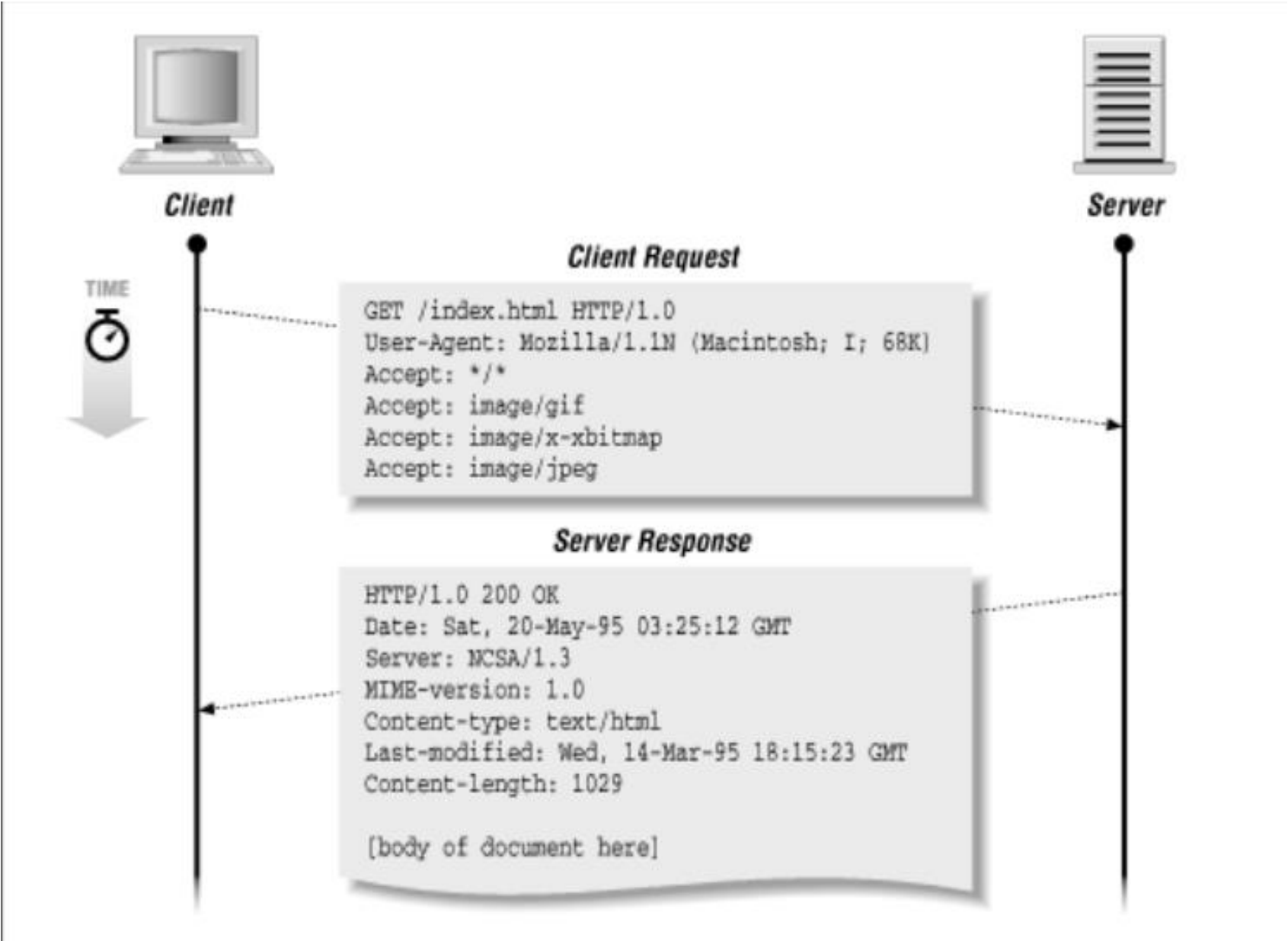


참고

<https://developer.mozilla.org/ko/docs/Web/HTTP/Status>



request message



REQUEST

GET http://127.0.0.1:5500/styles/navigation.css HTTP/1.1

HTTP request line

HOST: 127.0.0.1:5500  
Accept: text/css,\*/\*;q=0.1  
Accept-Language: en-GB,en;q=0.5  
Accept-Encoding: gzip, deflate, br  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101 Firefox/102.0  
Connection: keep-alive  
<CRLF>

HTTP headers

HTTP body (empty)

RESPONSE

HTTP/1.1 200 OK

HTTP response status line

Date: Wed, 06 Jul 2022 09:30:28 GMT  
Accept-Ranges: bytes  
Content-Length: 2005  
Content-Type: text/css; charset=UTF-8  
<CRLF>

HTTP response headers







nav.navbar {  
 ...some style  
}

HTTP response body



## 📌 request method

### HTTP Request Methods

 <b>GET</b>	 <b>POST</b>	 <b>PUT</b>	 <b>DELETE</b>	 <b>PATCH</b>	 <b>HEAD</b>
retrieve data from server	add data to an existing file or resource	update(replace) an existing file or resource in server	delete data from server	update a resource partially (modify)	retrieve the resource's headers

- **CONNECT** is used to open a two-way socket connection to the remote server;
- **OPTIONS** is used to describe the communication options for specified resource;
- **TRACE** is designed for diagnostic purposes during the development.
- **HEAD** retrieves the resource's headers, without the resource itself.

참고

<https://softuni.org/dev-concepts/everything-you-need-to-know-about-http-protocol/>



# 04 ajax



## ④ AJAX (Asynchronous Javascript and XML)

자바스크립트를 사용하여 브라우저가  
서버에게 비동기 방식으로 데이터를 요청하고,  
서버가 응답한 데이터를 수신하여 웹페이지를 동적으로 갱신하는 프로그래밍 방식

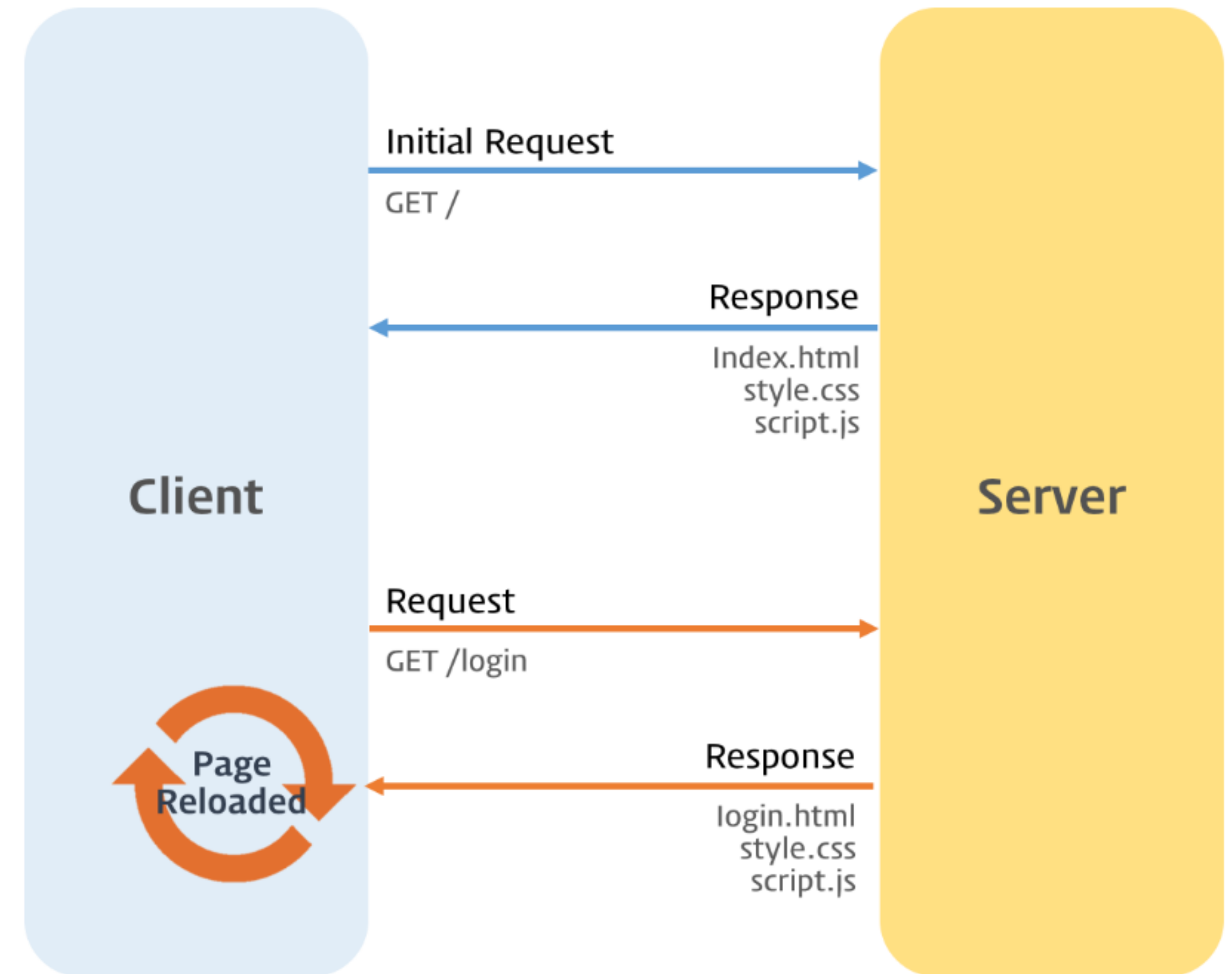
브라우저에서 제공하는 Web API인 XMLHttpRequest 객체를 기반으로 동작  
XMLHttpRequest 객체는 HTTP 비동기 통신을 위한 프로퍼티와 메서드를 제공



## 👍 AJAX 이전 방식

완전한 HTML을 서버로부터 전송 받아 웹 페이지 전체를 처음부터 다시 렌더링하는 방식으로 동작

화면이 전환되면 서버로부터 새로운 HTML을 전송 받아 웹 페이지 전체를 리렌더링



\* Request/Response는 파일 단위로 실시된다. 즉 html, css, js 파일은 한번에 Request/Response되는 것이 아니라 각각의 파일 단위로 Request/Response된다.

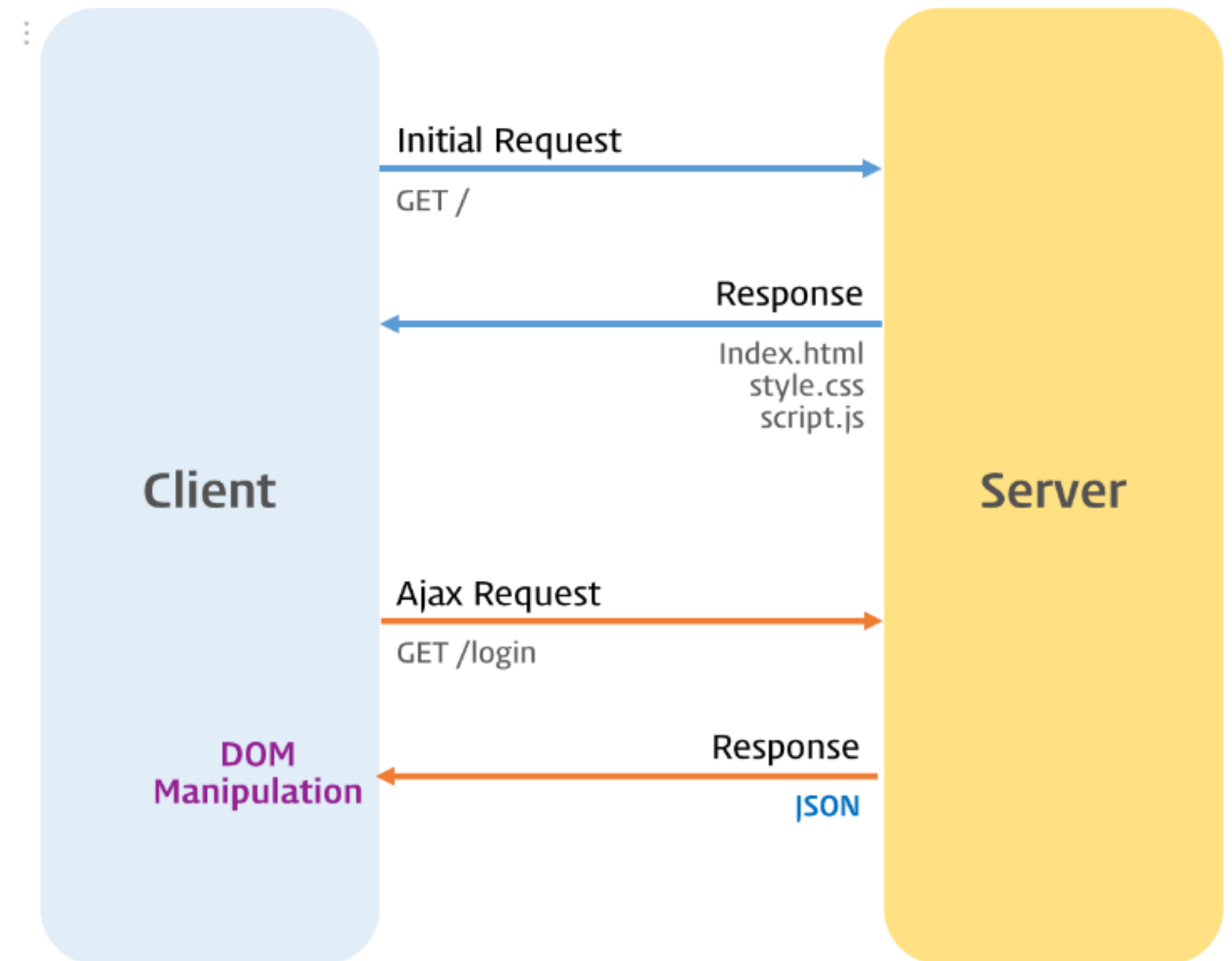


## ④ AJAX 모던 방식

필요한 데이터만 서버로부터 전송 받기 때문에 불필요한 데이터 통신이 발생하지 않는다.

변경할 필요가 없는 부분은 리렌더링 하지 않는다. 즉 화면의 깜박임 현상이 없다.

클라이언트 - 서버 간의 통신이 비동기 방식으로 작동, 따라서 서버에게 요청을 보낸 후 블로킹(blocking)이 발생하지 않는다.



\* Request/Response는 파일 단위로 실시된다. 즉 html, css, js 파일은 한번에 Request/Response되는 것이 아니라 각각의 파일 단위로 Request/Response된다.



## ✓ fetch API

fetch API는 브라우저 API 중 한 개로 네트워크 전송에 관한 비동기 통신 API  
다양한 주문 전송, 사용자 정보 읽기, 서버에서 데이터 받아옴 => 새로운 코딩 없이 데이터 가져옴

1. 서버에서 응답 헤더를 받자마자 fetch 호출 시 promise가 반환 됨  
[option] : 선택 매개변수(method나 header 지정가능)

```
fetch(url, [options])  
  .then((response) => response.json())  
  .catch((err) => console.log("err:", err))
```

참고 [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)



## ☑ option

method : 사용할 메소드를 선택 ('GET', 'POST', 'PUT', 'DELETE' )

headers : 헤더에 전달할 값 ( { 'content-Type': 'application/json' } )

body : 바디에 전달할 값 ( JSON.stringify(data) )

mode : 'cors' 등의 값을 설정 (cors, no-cors, same-origin)

credentials : 자격 증명을 위한 옵션 설정 (include, same-origin, omit)(default = same-origin)

cache : 캐쉬 사용 여부 (no-cache, reload, force-cache, only-if-cached)

옵션 값을 안주면 기본으로 GET이 들어가 있는 형태



## 📌 response 객체

response.status – 응답의 HTTP 코드

response.ok – 응답 상태가 200과 299 사이에 있는 경우 true 나머진 false

response.headers – 맵과 유사한 형태의 HTTP 헤더

response.text() – 응답을 텍스트 형태로 반환함

response.json() – 응답을 파싱해 JSON 객체로 변경함

response.formData() – 응답을 FormData 객체 형태로 반환

response.blob() – 응답을 Blob(타입이 있는 바이너리 데이터) 형태로 반환

method – HTTP 메서드

headers – 요청 헤드가 담긴 객체(제약 사항이 있음)

body – 보내려는 데이터(요청 본문)로 string이나 FormData, JSON, Blob 등 객체 형태