

삼성 청년 SW 아카데미

Javascript

<알림>

본 강의는 삼성 청년 SW아카데미의 콘텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

9장. 비동기 프로그래밍

챕터의 포인트

- 동기와 비동기
- Callback
- Promise
- async / await
- Thread / Async 비교

동기와 비동기

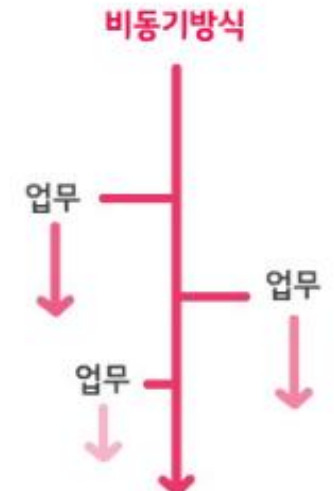
동기 방식

한 업무가 끝나야, 다음 업무를 시작한다.

비동기 방식

다른 업무를 기다리지 않고, 진행한다.

동시에 진행되는 것이 아니다!



화면에 출력될 코드를 예상해보자

```
1 console.log("1")  
2 console.log("2")  
3 console.log("3")
```

```
console.log("1")  
console.log("2")  
console.log("3")
```

1

2

3

우리가 기존에 작성했던 코드는 “동기”이다.

- 소스 코드의 흐름은 위에서 아래로, 좌에서 우로 진행된다.
- 순서가 보장되며, 위의 코드가 오래 걸리면 아래 코드는 진행되지 않는다.

비동기코드는 어떻게 동작할지 예상해보자.

- `setTimeout` : 비동기 함수로, 두번째 매개변수 시간(ms) 만큼 시간이 지난 후, 첫번째 매개변수를 실행

```
console.log('1')  
  
setTimeout(() => {  
    console.log('2')  
}, 1000);  
console.log('3');
```

1

3

2

앞으로 비동기에 대해서 배우게 되면

- 소스 코드의 흐름이 위에서 아래, 좌에서 우를 보장하지 않는다.
- 순서가 보장되지 않는 대신 오래 걸리는 작업 대신, 다른 작업을 먼저 진행할 수 있다.

친구와 함께 만나 영화보기로 했지만, 친구가 아직 도착하지 않는다면

• 동기방식 행동

- 계속 기다리다가, 친구 오면 영화 보러 간다.

친구가 도착해야, 다음 업무를 수행 가능

• 비동기방식

1. 도착하면 알려 달라고 전화한다.
2. 카페에서 다른 업무를 한다.
3. 친구에게 연락오면, 친구를 만나 영화를 본다.

친구가 도착하지 않아도, 다른 업무 수행 가능



sleep 함수 구현

- 시작 시간을 저장 해둬
- Time Limit 시간을 계산
- 현재 시간이 Time Limit 시간이 될때까지 무한 Loop

```
1  ✓ function sleep(delay){  
2      const start = Date.now();  
3      let limit = start + delay;  
4      while(Date.now() < limit){}  
5  }  
6  
7  console.log("빨래");  
8  sleep(3000);  
9  console.log("설거지");
```

3,000ms 동안 다른 작업을 수행할 수 없다.

setTimeout 함수

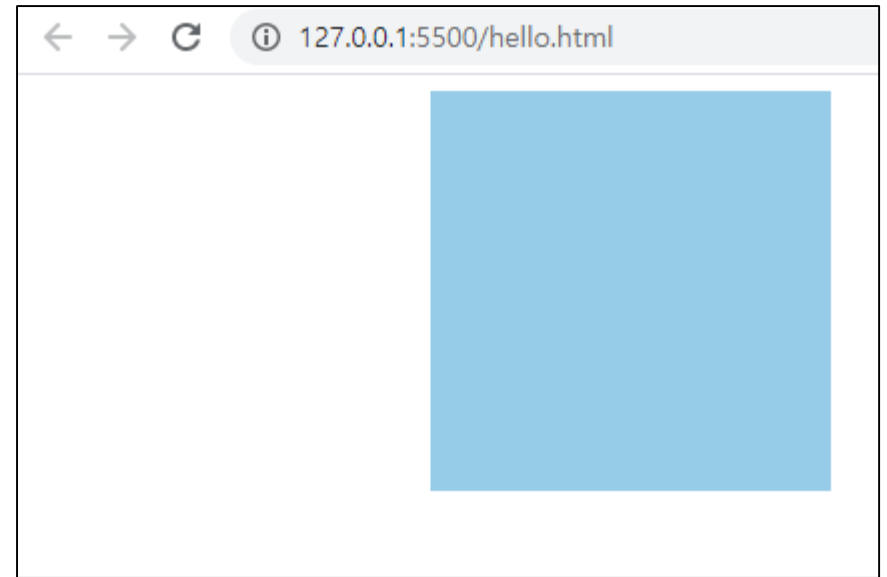
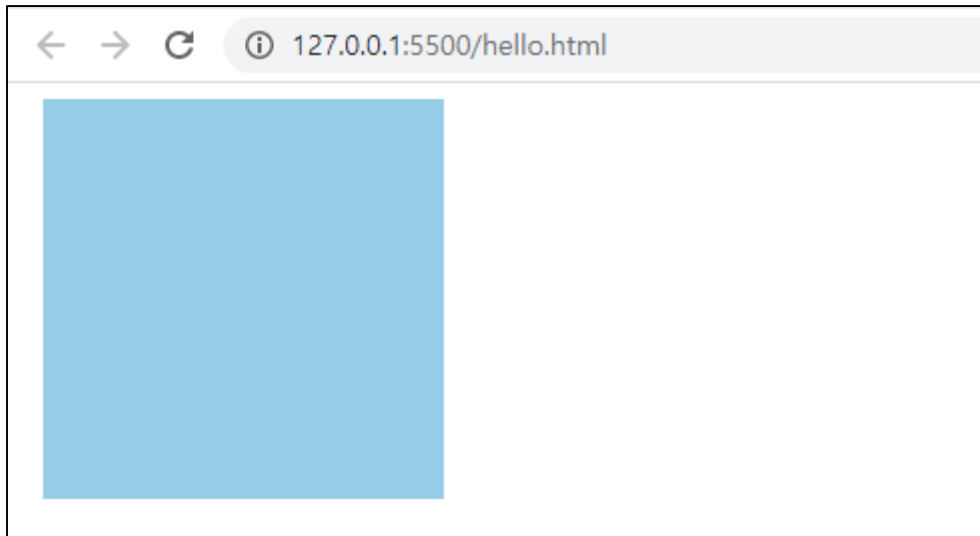
1. 콜백함수를 등록 해둬
2. 특정 시간이 지나면 콜백함수가 호출 됨

```
1  setTimeout(() => {  
2    console.log("빨래")  
3  }, 3000)  
4  
5  console.log("설거지")  
6  
7  // setTimeout(() => {  
8  //    console.log("설거지")  
9  //  }, 1000)
```

3,000ms 동안 다른 작업을
수행이 가능하다.

다음 동작이 되는 페이지 제작

- `width: 200px; height: 200px; background-color:skyblue;`
- `element.style.transform = "translate(200px)";`사용
- 클릭시 1.5초후 200px만큼 이동하는 방식 구현하기



Callback

setTimeout 비동기 함수를 학습하면서 첫번째 매개변수에 함수를 넣었다. 비슷한 방식을 이미 학습하였다.

- addEventListener의 두번째 매개변수에 콜백함수를 넣었다.

```
const body = document.querySelector('body');  
  
body.addEventListener('click', function(){  
    console.log("hi")  
});
```

‘등록’되는 함수를 Callback 함수라고 한다.

- 이벤트 발생시 호출되는 예약 함수

클릭 이벤트시 호출 될 함수를 등록한다. (= 콜백함수 등록)
그리고, 클릭 이벤트 발생시 등록된 콜백함수가 호출 된다.

- 일반 함수의 Parameter로 등록되는 함수

```
function run(e){  
    alert(e.code);  
}  
  
document.addEventListener('keyup', run);
```

run은 event의 callback 함수이다.

이벤트가 발생하면, 콜백함수가 호출되도록 한다.

- 비동기방식

1. 도착하면 알려달라고 전화한다.
2. 카페에서 다른 업무를 한다.
3. 친구에게 연락오면, 친구를 만나 영화를 본다.

콜백함수 등록
(아직 함수 호출하지 않는다)

이벤트 발생시
등록된 콜백함수 호출



다음 코드의 결과를 예상해보자.

```
setTimeout(() => {  
    console.log("첫번째 일")  
}, 5000)  
setTimeout(() => {  
    console.log("두번째 일")  
}, 3000)  
setTimeout(() => {  
    console.log("세번째 일")  
}, 1000)  
setTimeout(() => {  
    console.log("네번째 일")  
}, 2000)  
setTimeout(() => {  
    console.log("다섯번째 일")  
}, 4000)
```

- 순서가 보장이 안되서 어떤 일이 먼저 끝날지 보장할 수 없다

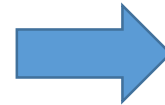
세번째 일	<u>test.js:8</u>
네번째 일	<u>test.js:11</u>
두번째 일	<u>test.js:5</u>
다섯번째 일	<u>test.js:14</u>
첫번째 일	<u>test.js:2</u>

- 그렇다면 비동기 함수로 순서를 보장 시킬 수 있을까?

비동기 함수의 순서 보장이 가능하다.

- 첫번째 비동기 함수의 뒤에, 두번째 비동기 함수를 넣으면 된다.

```
setTimeout(() => {  
  console.log("첫 번째 일")  
  setTimeout(() => {  
    console.log("두 번째 일")  
  }, 3000)  
, 5000)
```

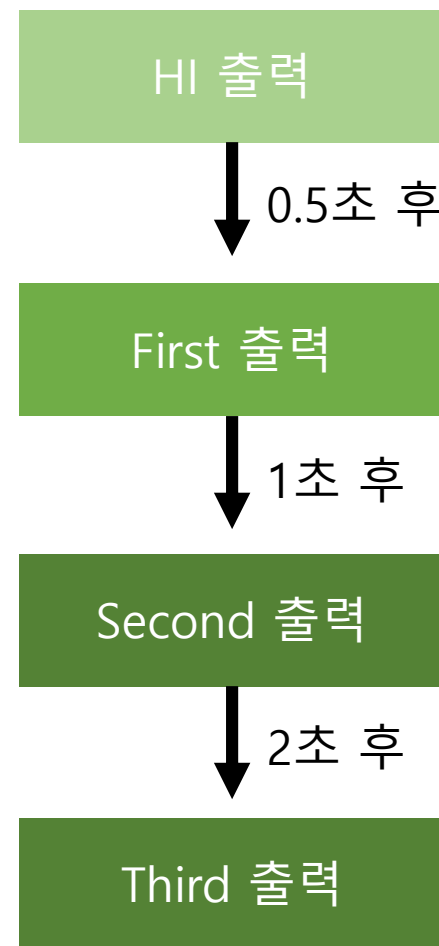


첫 번째 일
두 번째 일

setTimeout 을 사용하여 Sync(동기) 동작 구현

1. 웹페이지가 로딩 되자마자 console.log 에 HI 출력
2. 0.5 초 후 First가 콘솔에 출력
3. 1 초 후 Second가 콘솔에 출력
4. 2 초 후 Third가 콘솔에 출력

▶	⊘	top
HI		
First		
Second		
Third		
>		



다음 소스코드를 이해 해 보자.


- 비동기 함수로 동기 동작 구현하는 코드

```
1  <script>
2
3      console.log("HI");
4
5      setTimeout(()=>{
6          console.log("one");
7          setTimeout(() => {
8              console.log("two");
9              setTimeout(() => {
10                 console.log("three");
11                 setTimeout(() => {
12                     console.log("Four");
13                     setTimeout(() => {
14                         console.log("Five");
15                     }, 1000);
16                 }, 800);
17             }, 600);
18         }, 400);
19     }, 200);
20
21 </script>
22
```

Aync 함수들로 Sync 동작을 구현하는 코드 = Callback Hell

- 이해하기 힘들 정도로 가독성이 떨어져, 유지보수가 힘들다.

```
1 function hell(win) {  
2   // for listener purpose  
3   return function() {  
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {  
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {  
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {  
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {  
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {  
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {  
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {  
11                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {  
12                  loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {  
13                   async.eachSeries(SERIALS, function(src, callback) {  
14                     loadScript(win, BASE_URL+src, callback);  
15                   });  
16                 });  
17               });  
18             });  
19           });  
20         });  
21       });  
22     });  
23   });  
24 });  
25 };  
26 }
```




Sync

특정 동작이 끝난 이후, 다른 동작을 수행하기 시작 함

Async(Asynchronous)

1. Callback 함수 등록 처리 (호출하지 않음)
2. 다른 동작 동작을 수행
3. 특정 이벤트 발생시, Callback 함수 수행 됨

내부적으로
다른 동작을 수행하면서도,
지속적으로 Event가 발생했는지 감시



Promise

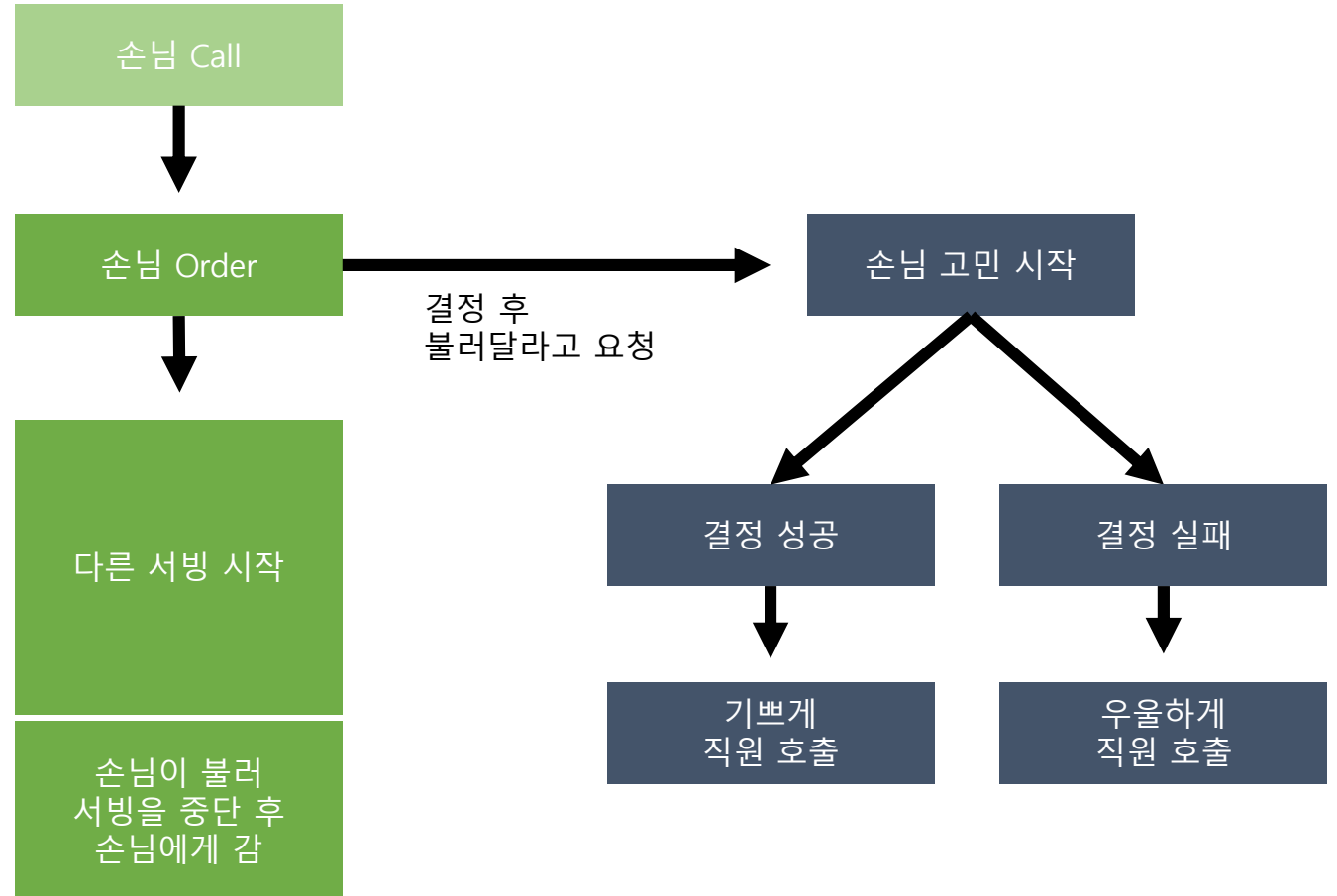
Promise

- 비동기 작업을 좀 더 편하게 할 수 있도록 ES6에서 도입
- 콜백 방식의 콜백지옥을 보완
- `new Promise((resolve, reject) => { });`
- **pending 상태**
 - 대기 상태, resolve나 reject가 실행되기 전의 상태
- **fulfilled 상태**
 - 이행 상태, resolve 호출 시 의 상태
- **rejected 상태**
 - 실패 상태, reject 호출 시의 상태

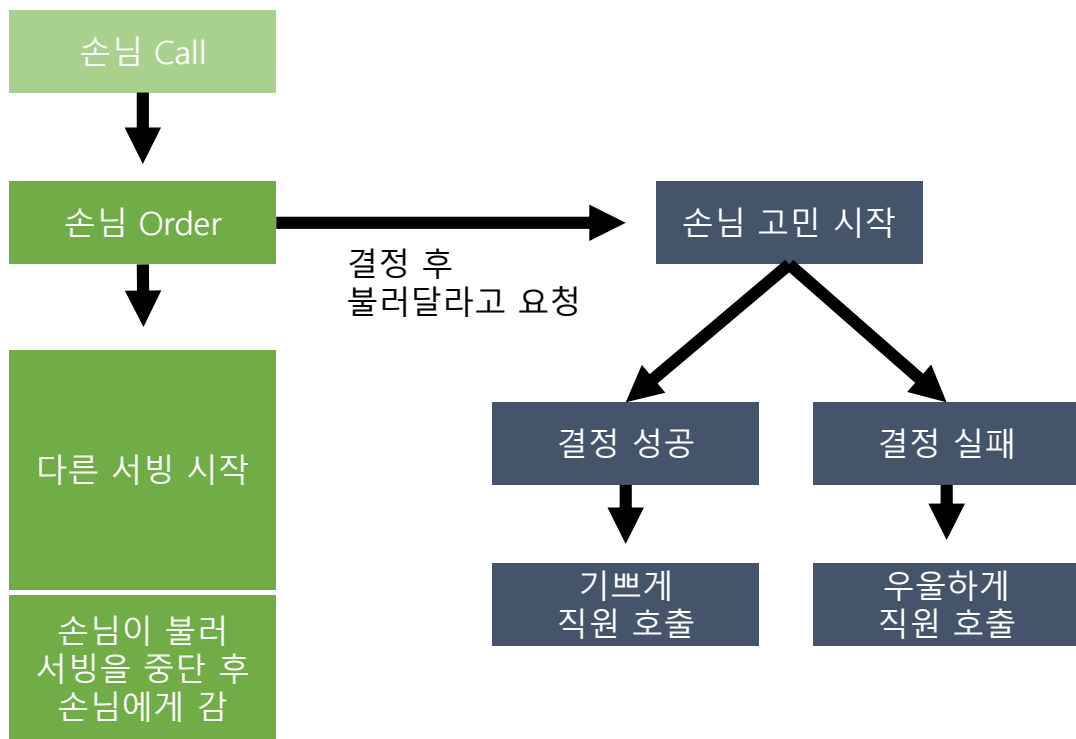
The logo for Promise JS, featuring the word "Promise" in a light grey sans-serif font above the letters "JS" in a large, bold, black sans-serif font. The "JS" is positioned on a yellow rectangular background that occupies the left half of the logo's container.

카페 점원 시나리오

- 손님이 불러 서빙을 중단하고, 손님의 결정에 따라 대처해보자.
- 손님이 결정에 성공했다면?
→ GOOD 출력
- 손님이 결정을 하지 못했다면?
→ OH NO 출력



resolve를 호출하면 GOOD 출력 / reject를 호출하면 OH NO 출력



```
1 <script>
2
3 console.log("어서오세요");
4 console.log("주문하십시오");
5 console.log("고민하는동안 다른 것을 하고 있겠소");
6 const pro = new Promise((resolve, reject) => {
7   setTimeout(() => {
8     console.log("나 이제 주문하겠소");
9     resolve();
10    //reject();
11   }, 1000)
12 });
13 console.log("서빙 서빙 서빙");
14
15 pro.then(() => {
16   console.log("GOOD");
17 }).catch(() => {
18   console.log("OH NO");
19 });
20
21
22 </script>
```

Promise Chaining

- `Promise.then`을 호출하면 자동적으로 Promise가 return 된다

```
Promise.resolve().  
  then(onFulfilled1).  
  then(onFulfilled2).  
  then(onFulfilled3).  
  catch(onRejected);
```


Promise Chaining을 활용한 덧셈 만들기

- promise chain을 활용해서 누적으로 값을 1부터 5까지 더해보자

```
1  function plus() {  
2    return new Promise(function (resolve, reject) {  
3      resolve(0);  
4    });  
5  }  
6  
7  plus()  
8    .then((sum) => {  
9      return sum + 1;  
10   })  
11   .then((sum) => sum + 2)  
12   .then((sum) => sum + 3)  
13   .then((sum) => sum + 4)  
14   .then((sum) => sum + 5)  
15   .then((sum) => {  
16     console.log(sum);  
17   });  
18
```

햄버거 만들기

- Promise Chaining을 활용해서
- 3초 후에 참깨 빵
- 순 쇠고기 패티 2장
- 특별한 소스
- 양상추
- 치즈 피클 양파
- 순서로 햄버거를 쌓아보자

참깨빵위에

참깨빵위에 순쇠고기 패티 2장

참깨빵위에 순쇠고기 패티 2장 특별한소스

참깨빵위에 순쇠고기 패티 2장 특별한소스 양상추

참깨빵위에 순쇠고기 패티 2장 특별한소스 양상추 치즈 피클 양파까지

async / await

다음 코드의 결과를 예상해보자.

- Axios 뒤부분은, 비동기 통신때 배울 코드이며, 서버의 데이터를 받아온다

```
<body>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script>
    function getData(){
      return new Promise((resolve, reject) => {
        resolve(axios.get("https://jsonplaceholder.typicode.com/todos"))
      })
    }
    console.log(getData())
  </script>
</body>
```

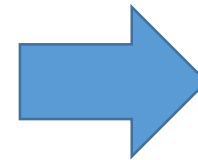
잘 받아오는 듯 하다.. 하지만 필요한 값은 data안에 있다

```
▼ Promise {<pending>} ⓘ  
  ► [[Prototype]]: Promise  
    [[PromiseState]]: "fulfilled"  
  ▼ [[PromiseResult]]: Object  
    ► config: {transitional: {...}, transformRequest: Array(1), transformResponse: Array(1), timeout: 0  
    ► data: (200) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}  
    ► headers: {cache-control: 'max-age=43200', content-type: 'application/json; charset=utf-8', expires: 0  
    ► request: XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0, withCredentials: false  
      status: 200  
      statusText: ""  
    ► [[Prototype]]: Object
```

getData함수의 data에 접근해보면 정상적으로 출력되지 않는다

- 왜 이런 현상이 발생하는 것일까?

```
<body>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script>
    function getData(){
      return new Promise((resolve, reject) => {
        resolve(axios.get("https://jsonplaceholder.typicode.com/todos"))
      })
    }
    console.log(getData().data)
  </script>
</body>
```




undefined

- Then 후속처리 메서드를 통해서 접근해보자.

[illegible]

- 이유는 then 처리는 `getData`의 결과가 발생한 '후'에 동작하기 때문에 위치가 중요하지 않다.

[illegible]

코드가 많아지면 많아 질수록, 소스 코드의 흐름을 파악하기 어렵다

- 해결 하기위해 `async/await`가 출현
- 소스 코드의 흐름이 위에서 아래로 (동기적)으로 보이며, 정상적으로 접근할 수 있다.

```
✓ <body>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script>
    (async function getData(){
      const result = await axios.get("https://jsonplaceholder.typicode.com/todos")
      console.log(result.data)
    })()
  </script>
</body>
```

화면에 출력하기

- `axios.get("https://api.thedogapi.com/v1/images/search")`로 axios 통신하여
- 만들어놓은 `img` 태그에 `src`경로에 할당한다
- 요청 정보는 `data`안에 있다.

```
<body>
  <img src="">
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script>
    function getData(){
    }
    getData()
  </script>
</body>
```



async/await

- Promise를 쉽게 사용하기 위해 ES8에 도입
- async는 함수 앞에 위치.
- await는 async 함수 내부에 위치.
- promise를 반환한다
- 비동기 코드를 동기적으로 작성한다

```
1  function eat() {
2    return new Promise(function (resolve, reject) {
3      setTimeout(function () {
4        resolve("먹어!!");
5      }, 2000);
6    });
7  }
8
9  async function wait() {
10   console.log("기다려!!!!");
11   const data = await eat();
12   console.log(data);
13   console.log("잘했어!!");
14 }
15 wait();
16
```

async/await를 활용한 햄버거 재료 나타내기

- 1초 후에 참깨빵위에
- 2초 후에 순쇠고기 패티 2장
- 3초후에 특별한 소스
- 4초후에 양상추
- 5초후 치즈 피클 양파를 각각 console.log로 나타내기

참깨빵 위에	<u>VM607:7</u>
순쇠고기 패티2장	<u>VM607:8</u>
특별한 소스	<u>VM607:9</u>
양상추	<u>VM607:10</u>
치즈 피클 양파까지	<u>VM607:11</u>

Thread / Async 차이

Single Thread vs Multi Thread

Async(비동기) 프로그래밍과 Thread 프로그래밍의 차이를 정확히 이해 해 보자.

Single Thread 지점 카페 아르바이트생

- 아르바이트 교육 받은 내용
 1. 주문을 받는다.
 2. 커피가 나올 때 까지 기다린다.
 3. 커피를 받는다.
- 손님이 커피를 받기 전에 다른 사람이 온다면?
 - 줄을 선다.



Multi Thread 지점 카페 아르바이트생

- 종업원이 늘어났다
 - 각자 맡고 있는 파트가 생겨났다.
 - 주문 파트
 - 요리 파트
- 고려할 부분
 - 종업원들의 정보 전달 및 동선 관리가 중요하다

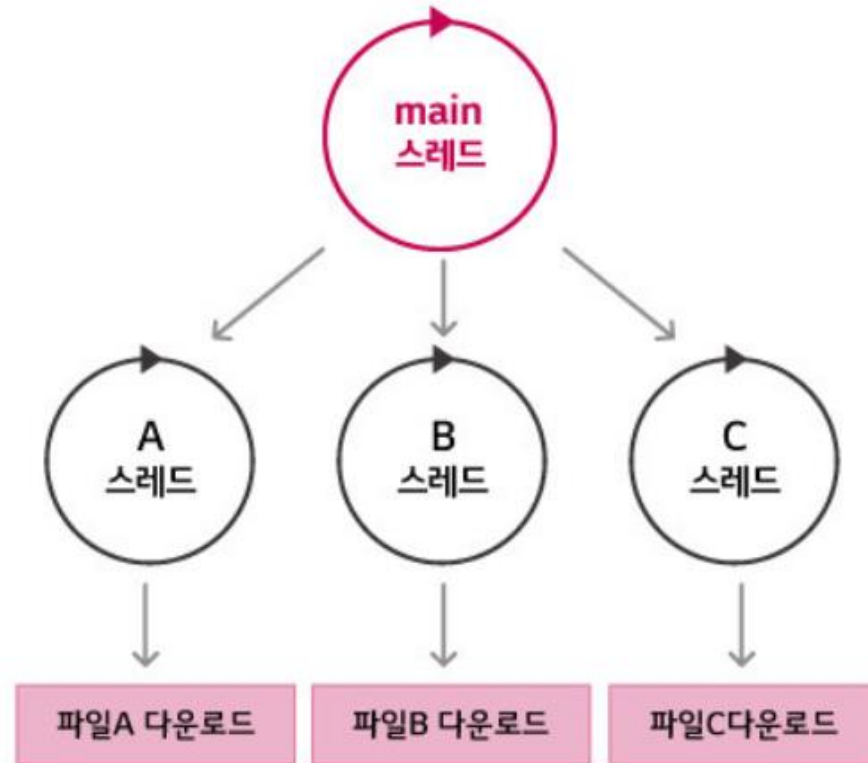


싱글 스레드



순차 실행(Sequential)

멀티 스레드



병행 실행(Concurrent)

Async 지점 카페 아르바이트생

- 주문을 한다
- 주문 완료 시 진동벨을 건네 받는다.
- 음료 제작 완료 시 진동벨로 호출 받는다.



Multi Thread Programming (멀티 쓰레드 프로그래밍)

- 여러 함수들을 동시에 수행할 수 있다.

Async Programming (비동기 프로그래밍)

- 시간이 걸리는 함수를 잠시 보류하고, 다른 함수 부터 수행한다.

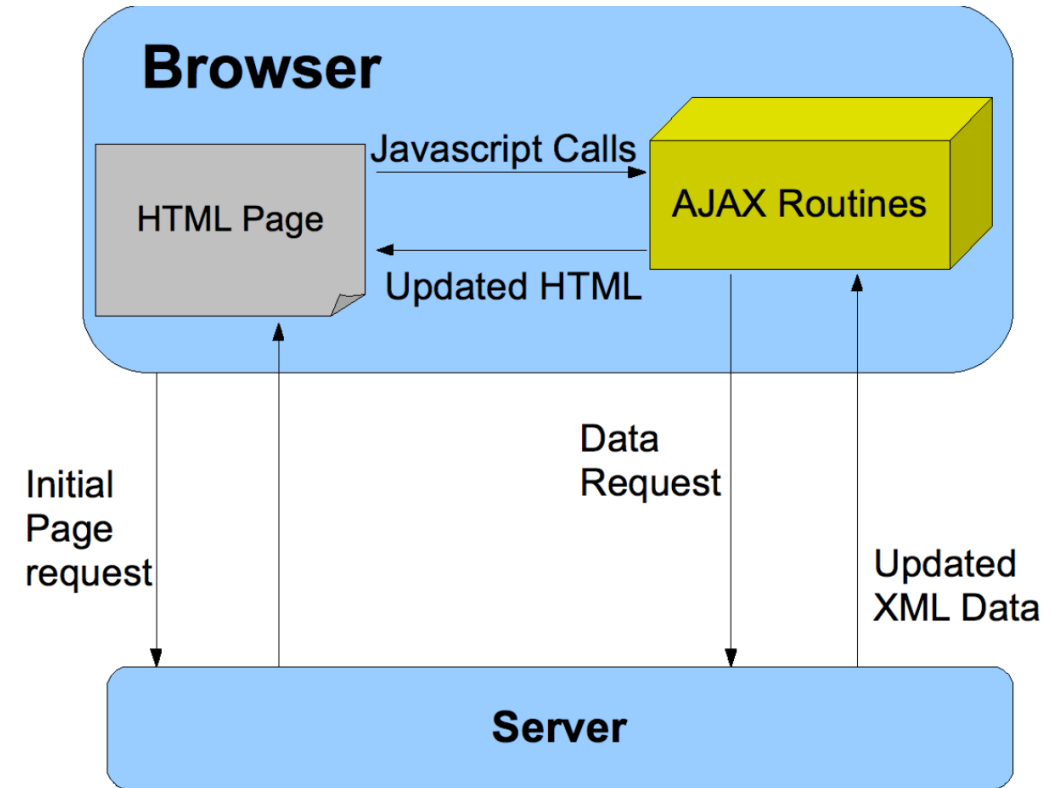
10장. Ajax

챕터의 포인트

- Ajax개요
- XMLHttpRequest
- fetch
- axios

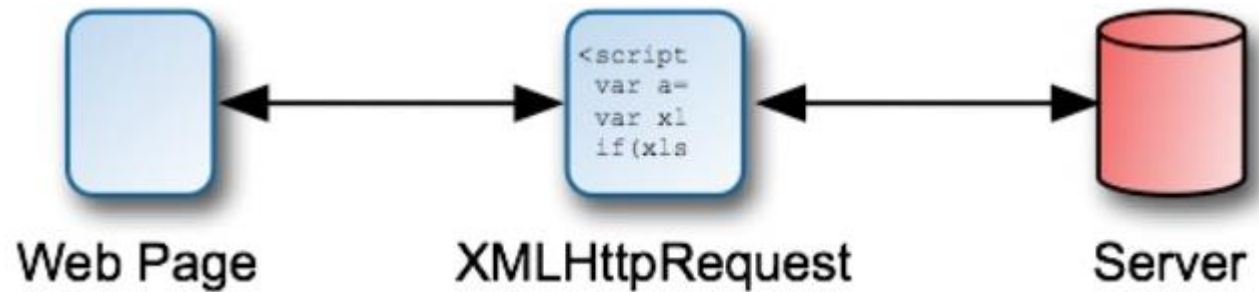
Ajax(Asynchronous Javascript and XML)

- 서버와 통신하기 위해 XMLHttpRequest 객체 사용
- 비동기성으로 페이지 새로고침을 하지않고도 수행된다.



XMLHttpRequest 를 활용해 Ajax 통신해보기

- XMLHttpRequest를 통해 비동기 Ajax통신의 기초를 익힌다



fetch API

- XMLHttpRequest와 유사하나 더 발전된 API (es6 도입)
- 객체를 Promise 형태로 반환 받는다.
- json의 타입별로 쉽게 적용이 가능하다.

Get started with
Fetch API

가장 널리 쓰이는 http 통신 라이브러리

- Vue/React 에서도 권고 라이브러리로 axios가 지정이 되어 있다.
- Promise 형태 리턴
- 사용하기 편리하다.

axios

0.19.2 • Public • Published 6 months ago

[Readme](#)
[Explore](#) BETA
[1 Dependency](#)
[39,486 Dependents](#)
[43 Versions](#)

axios

npm **v0.19.2**
 build **passing**
 coverage **94%**
 Install size **419 kB**
 downloads **50M/month**
 chat **on gitter**

code helpers **139**

Promise based HTTP client for the browser and node.js

Features

- Make **XMLHttpRequests** from the browser

Install

```
> npm i axios
```

Weekly Downloads

17,034,972

Version

0.19.2

License

MIT

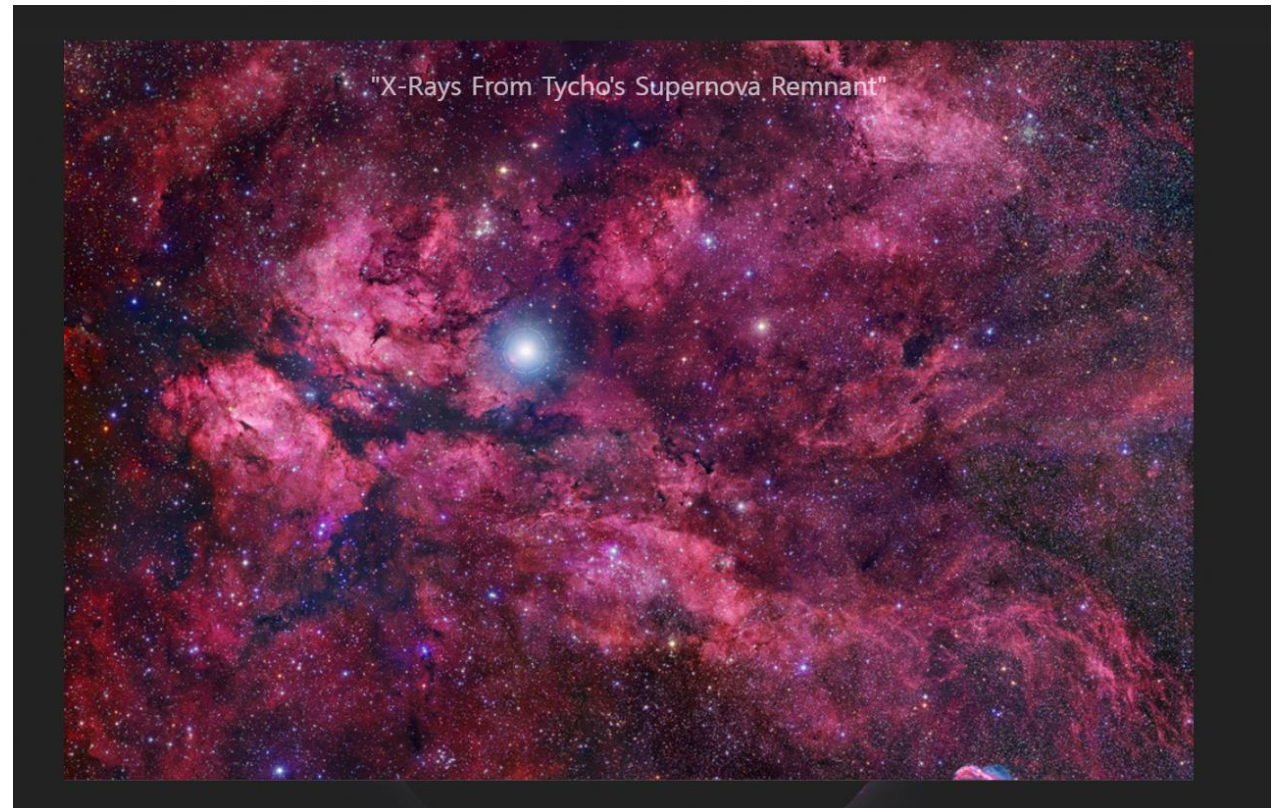
axios 호출값 받아와서 활용하기

- <https://jsonplaceholder.typicode.com/todos> 값을 가져와 배열에 담기
- 위의 data값을 받아 온 데이터에서
completed: false, userId 가 짝수인 경우,
title의 글자수가 10자 이상, title에 qua가 포함 되어있는 경우
배열에 담아 console 출력하기

```
▼ (19) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}] ①
  ▶ 0: {userId: 2, id: 23, title: "et itaque necessitatibus maxime molestiae qui quas velit", completed: fals...
  ▶ 1: {userId: 2, id: 24, title: "adipisci non ad dicta qui amet quaerat doloribus ea", completed: false}
  ▶ 2: {userId: 4, id: 62, title: "et placeat et tempore aspernatur sint numquam", completed: false}
  ▶ 3: {userId: 4, id: 64, title: "voluptas consequatur qui ut quia magnam nemo esse", completed: false}
  ▶ 4: {userId: 4, id: 69, title: "doloribus sint dolorum ab adipisci itaque dignissimos aliquam suscipit", c...
  ▶ 5: {userId: 4, id: 78, title: "reiciendis est magnam amet nemo iste recusandae impedit quaerat", complete...
  ▶ 6: {userId: 6, id: 112, title: "consectetur impedit quisquam qui deserunt non rerum consequuntur eius", c...
  ▶ 7: {userId: 6, id: 113, title: "quia atque aliquam sunt impedit voluptatum rerum assumenda nisi", complet...
  ▶ 8: {userId: 6, id: 114, title: "cupiditate quos possimus corporis quisquam exercitationem beatae", comple...
  ▶ 9: {userId: 6, id: 117, title: "sequi quae est et qui qui eveniet asperiores", completed: false}
  ▶ 10: {userId: 6, id: 118, title: "quia modi consequatur vero fugiat", completed: false}
  ▶ 11: {userId: 6, id: 120, title: "dolorem laboriosam vel voluptas et aliquam quasi", completed: false}
  ▶ 12: {userId: 8, id: 145, title: "beatae qui ullam incidunt voluptatem non nisi aliquam", completed: false}
  ▶ 13: {userId: 8, id: 148, title: "esse quas et quo quasi exercitationem", completed: false}
  ▶ 14: {userId: 8, id: 153, title: "ea aperiam consequatur qui repellat eos", completed: false}
  ▶ 15: {userId: 8, id: 160, title: "et praesentium aliquam est", completed: false}
  ▶ 16: {userId: 10, id: 181, title: "ut cupiditate sequi aliquam fuga maiores", completed: false}
  ▶ 17: {userId: 10, id: 185, title: "voluptates dignissimos sed doloribus animi quaerat aut", completed: fal...
  ▶ 18: {userId: 10, id: 192, title: "ut quas possimus exercitationem sint voluptates", completed: false}
    length: 19
    proto : Array(0)
```

<https://api.nasa.gov/>

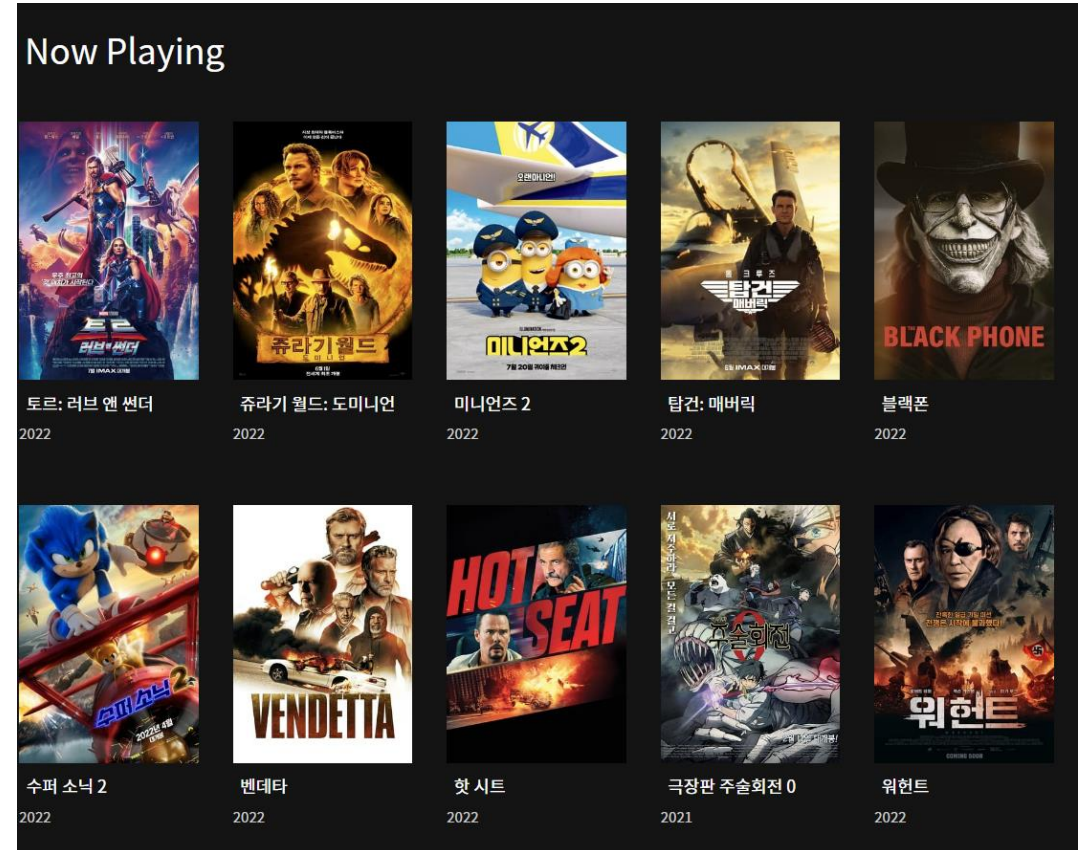
- API 발급
- https://api.nasa.gov/planetary/apod?api_key=발급한APIKEY&count=20
 - 요청값을 기반으로 프로젝트 진행하기



<https://www.themoviedb.org/>

- 가입하기 및 API 발급
- <https://developers.themoviedb.org/3>
- 데이터를 가공해서 본인만의 프로젝트 만들기

```
const request = axios.get("https://api.themoviedb.org/3/movie/now_playing", {  
  params: {  
    api_key: "발급받은 API KEY",  
    language: "ko-KR",  
  }  
})
```



내일 방송에서 만나요!

삼성 청년 SW 아카데미