

JavaScript

INDEX

- 동기와 비동기
- JavaScript의 비동기 처리
- Axios 라이브러리
- Callback과 Promise
- AJAX

동기와 비동기

| 개요

- JavaScript에서의 비동기 처리 학습

INTRO (1/4)

- 주문 후 커피가 나올 때까지 기다려야 함 (동기식)



INTRO (2/4)

- 주문 후 진동벨이 울리면 커피를 가져옴 (비동기)



INTRO (3/4)

<오늘의 할 일>

1. 교수님께 이메일 보내기
 - 교수님 답장 확인하기
 - 교수님께 답장 보내기
2. 저녁 식사 하기
3. 부모님께 안부 전화하기
 - 전화 통화가 안되면 문자로 안부 물어보기
4. 꿀잠자기

만약 위에서부터 **순서대로만** 처리 해야 한다면?

INTRO (4/4)

<오늘의 할 일>

1. 교수님께 이메일 보내기
 - 교수님 답장 확인하기
 - 교수님께 답장 보내기
2. 저녁 식사 하기
3. 부모님께 안부 전화하기
 - 전화 통화가 안되면 문자로 안부 물어보기
4. 꿀잠자기

만약 위에서부터 **순서대로만** 처리 해야 한다면?

교수님의 답장이 오지 않으면,
영원히 저녁 식사를 할 수 없다!



동기(Synchronous)

| 동기(Synchronous)

- 모든 일을 **순서대로 하나씩** 처리하는 것
- 순서대로 처리한다 == 이전 작업이 끝나면 다음 작업을 시작한다
- 우리가 작성했던 Python 코드가 모두 동기식

```
print('첫번째 작업')
for i in range(10):
    print('두번째 작업')
    print(i)
print('세번째 작업')
```

- 요청과 응답을 동기식으로 처리한다면?

요청을 보내고 응답이 올때까지 기다렸다가 다음 로직을 처리

웹에서의 동기 경험하기

```
<body>
  <button>버튼</button>
  <script>
    const btn = document.querySelector('button')
    btn.addEventListener('click', () => {
      alert('you clicked me!')
      const pElem = document.createElement('p')
      pElem.innerText = 'p Element'
      document.body.appendChild(pElem)
    })
  </script>
</body>
```

버튼

이 페이지 내용:
you clicked me!

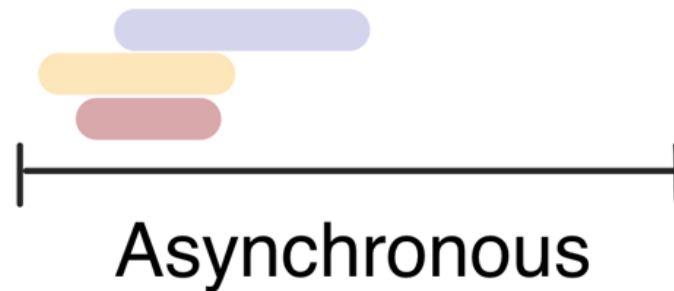
확인

확인을 누르기전까지 p태그는 보이지 않음

비동기(Asynchronous)

| 비동기(Asynchronous)

- 작업을 시작한 후 **결과를 기다리지 않고** 다음 작업을 처리하는 것 (병렬적 수행)
- 시간이 필요한 작업들은 요청을 보낸 뒤 응답이 빨리 오는 작업부터 처리
- 예시) Gmail에서 메일 전송을 누르면 목록 화면으로 전환되지만 실제로 메일을 보내는 작업은 병렬적으로 뒤에서 처리됨



비동기(Asynchronous)

```
function slowRequest(callback) {  
  console.log('1. 오래 걸리는 작업 시작 ...')  
  setTimeout(function () {  
    callback()    << 3초를 기다리는 함수 (오래 걸리는 작업)  
  }, 3000)  
}  
  
function myCallback() {  
  console.log('2. 콜백함수 실행됨') << 가장 마지막에 출력  
}  
  
slowRequest(myCallback)  
console.log('3. 다른 작업 실행')  
  
// 출력결과  
// 1. 오래 걸리는 작업 시작 ...  
// 3. 다른 작업 실행  
// 2. 콜백함수 실행됨
```

| 비동기(Asynchronous)를 사용하는 이유

- 사용자 경험


- 예를 들어 아주 큰 데이터를 불러온 뒤 실행되는 앱이 있을 때, 동기로 처리한다면 데이터를 모두 불러온 뒤에야 앱의 실행 로직이 수행되므로 사용자들은 마치 앱이 멈춘 것과 같은 경험을 겪게 됨
 - 즉, 동기식 처리는 특정 로직이 실행되는 동안 다른 로직 실행을 차단하기 때문에 마치 프로그램이 응답하지 않는 듯한 사용자 경험을 만들게 됨
 - 비동기로 처리한다면 먼저 처리되는 부분부터 보여줄 수 있으므로, 사용자 경험에 긍정적인 효과를 볼 수 있음
- 이와 같은 이유로 많은 웹 기능은 비동기 로직을 사용해서 구현되어 있음

이어서 ..

삼성 청년 SW 아카데미

JavaScript의 비동기 처리

Single Thread 언어, JavaScript

- 그러면 응답이 먼저 오는 순서대로 처리하지 말고, 아예 여러 작업을 동시에 처리하면 되지 않을까? 
 - JavaScript는 한 번에 하나의 일만 수행할 수 있는 **Single Thread** 언어로 동시에 여러 작업을 처리할 수 없음
- [참고] Thread란?
작업을 처리할 때 실제로 작업을 수행하는 주체로, multi-thread라면 업무를 수행할 수 있는 주체가 여러 개라는 의미
- 즉, JavaScript는 하나의 작업을 요청한 순서대로 처리할 수 밖에 없다!
그러면 어떻게 Single Thread인 JavaScript가 비동기 처리를 할 수 있을까?

JavaScript Runtime

- JavaScript 자체는 Single Thread이므로 비동기 처리를 할 수 있도록 도와주는 환경이 필요함
- 특정 언어가 동작할 수 있는 환경을 "런타임(Runtime)"이라 함
- JavaScript에서 비동기와 관련한 작업은 브라우저 또는 Node 환경에서 처리
- 이중에서 브라우저 환경에서의 비동기 동작은 크게 아래의 요소들로 구성됨
 1. JavaScript Engine의 Call Stack
 2. Web API
 3. Task Queue
 4. Event Loop

| 비동기 처리 동작 방식

- 브라우저 환경에서의 JavaScript의 비동기는 아래와 같이 처리된다.
- 1. 모든 작업은 **Call Stack**(LIFO)으로 들어간 후 처리된다.
- 2. 오래 걸리는 작업이 **Call Stack**으로 들어오면 **Web API**로 보내 별도로 처리하도록 한다.
- 3. Web API에서 처리가 끝난 작업들은 곧바로 **Call Stack**으로 들어가지 못하고 **Task Queue**(FIFO)에 순서대로 들어간다.
- 4. **Event Loop**가 **Call Stack**이 비어 있는 것을 계속 체크하고 **Call Stack**이 빈다면 **Task Queue**에서 가장 오래된 (가장 앞에 있는) 작업을 **Call Stack**으로 보낸다.

| 비동기 처리 동작 요소 (1/2)

1. Call Stack

- 요청이 들어올 때 마다 순차적으로 처리하는 Stack(LIFO)
- 기본적인 JavaScript의 Single Thread 작업 처리

2. Web API

- JavaScript 엔진이 아닌 브라우저에서 제공하는 runtime 환경
- 시간이 소요되는 작업을 처리 (setTimeout, DOM Event, AJAX 요청 등)

| 비동기 처리 동작 요소 (2/2)

3. Task Queue

- 비동기 처리된 Callback 함수가 대기하는 Queue(FIFO)

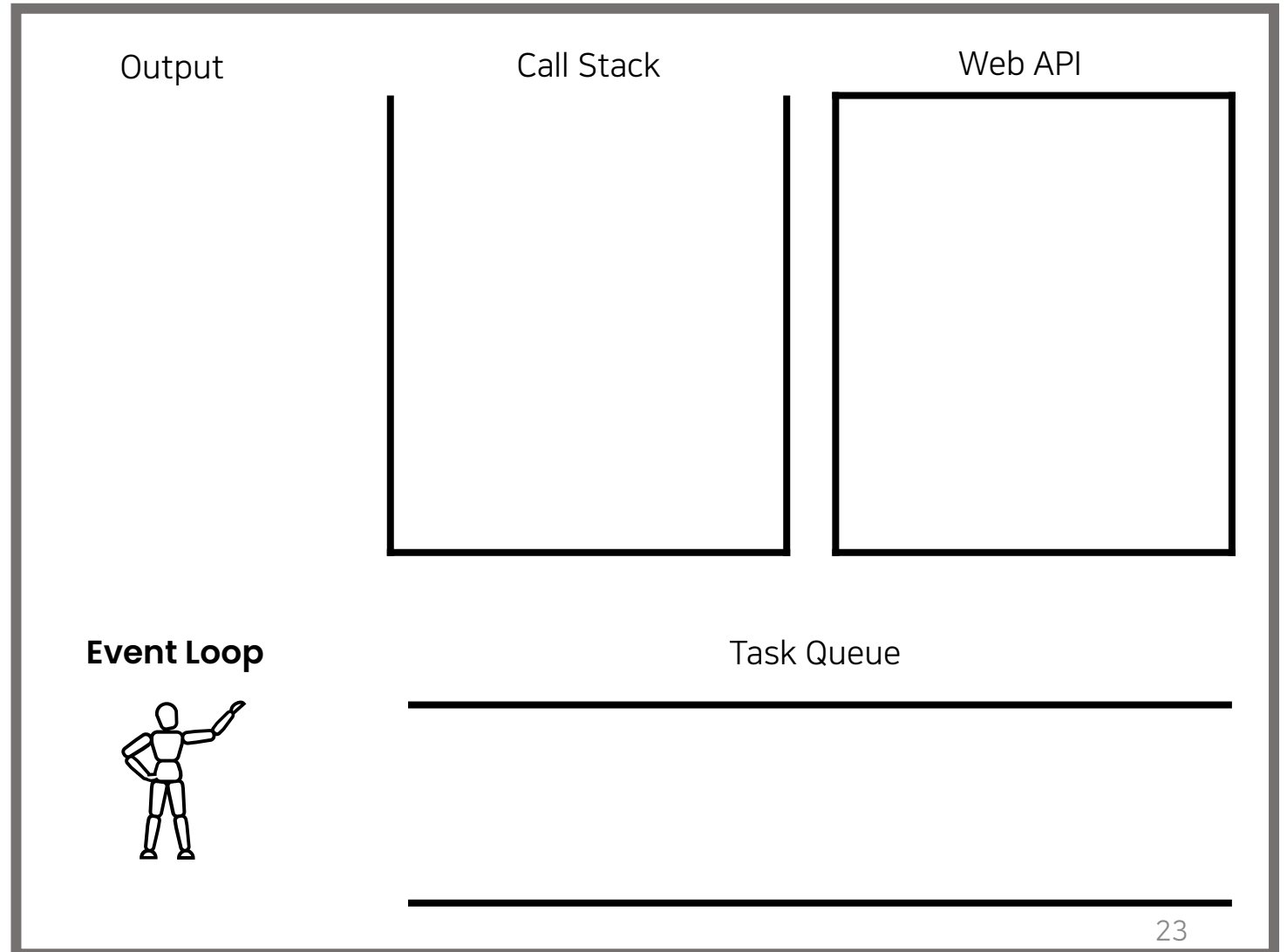
4. Event Loop

- Call Stack과 Task Queue를 지속적으로 모니터링
- Call Stack이 비어 있는지 확인 후 비어 있다면
Task Queue에서 대기 중인 오래된 작업을 Call Stack으로 Push

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (1/19)

```
console.log('Hi')  
  
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)  
  
console.log('Bye')
```



JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (2/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

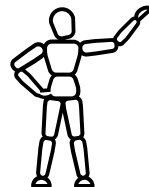
```
console.log('Bye')
```

Output

Call Stack

Web API

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (3/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

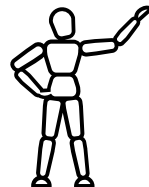
Output

Call Stack

Web API

```
console.log('Hi')
```

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (4/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

Output

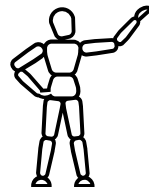
Hi

Call Stack

console.log('Hi')

Web API

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (5/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

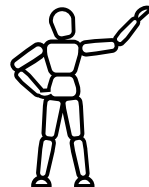
Output

Hi

Call Stack

Web API

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (6/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

Output

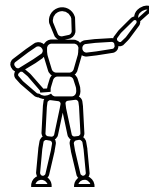
Hi

Call Stack

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

Web API

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (7/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

Output

Hi

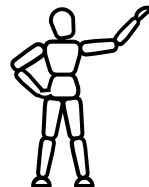
Call Stack

Web API

1초 경과

```
function ssafy() {  
  console.log('SSAFY')  
}
```

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (8/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

Output

Hi

Call Stack

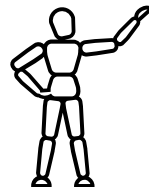
console.log('Bye')

Web API

2초 경과

```
function ssafy() {  
  console.log('SSAFY')  
}
```

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (9/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

Output

Hi

Bye

Call Stack



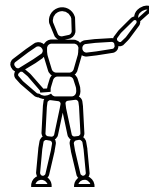
console.log('Bye')

Web API

2.5초 경과

```
function ssafy() {  
  console.log('SSAFY')  
}
```

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (10/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

Output

Hi

Bye

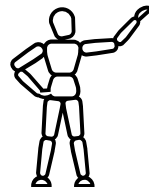
Call Stack

Web API

3초 경과

```
function ssafy() {  
  console.log('SSAFY')  
}
```

Event Loop



Task Queue



JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (11/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

Output

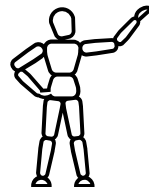
Hi

Bye

Call Stack

Web API

Event Loop



Task Queue

```
function ssafy() {  
  console.log('SSAFY')  
}
```

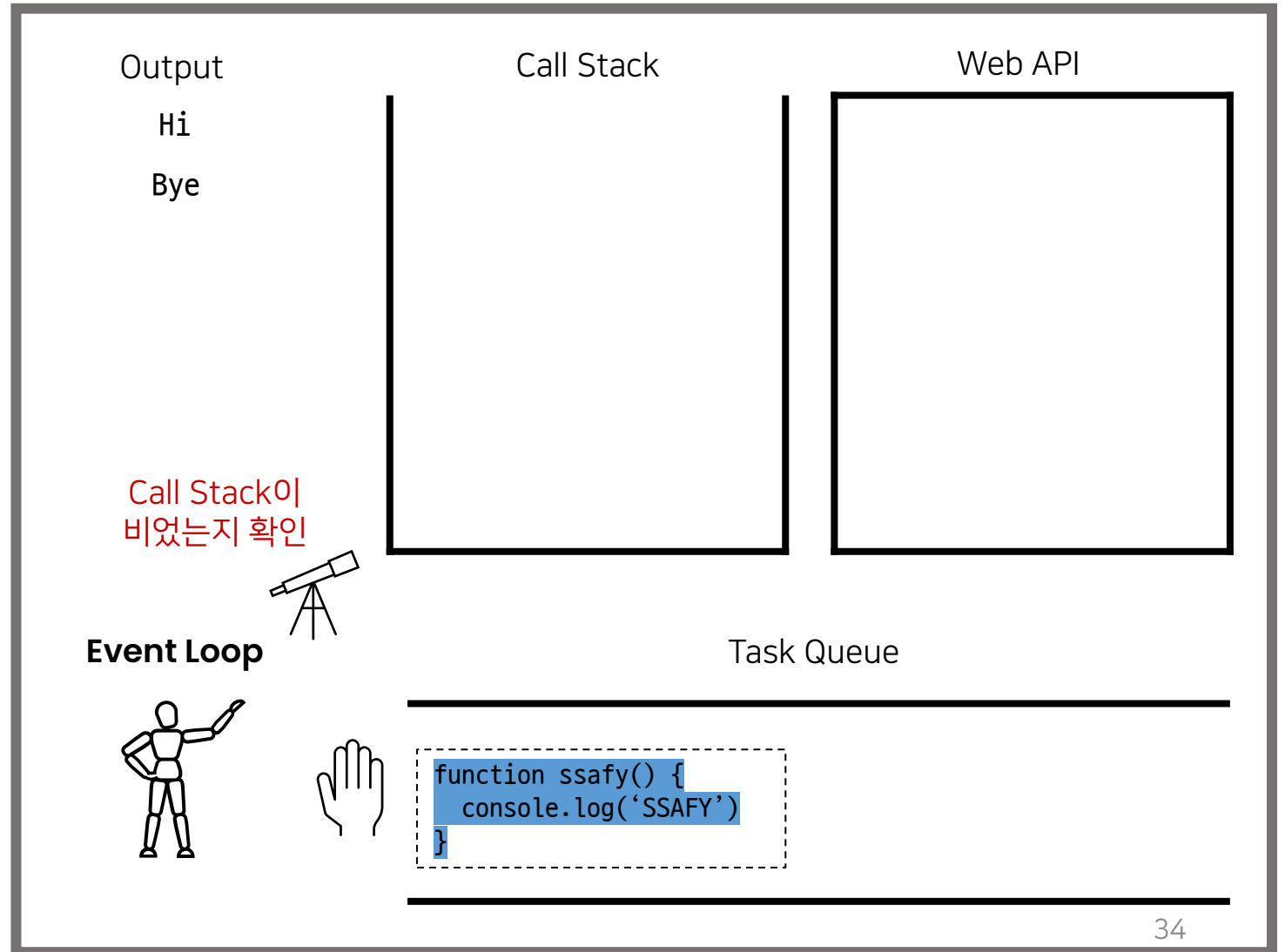
JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (12/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```



JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (13/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

Output

Hi

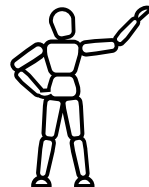
Bye

Call Stack

ssafy ()

Web API

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (14/19)

```
console.log('Hi')  
  
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)  
  
console.log('Bye')
```

Output

Hi

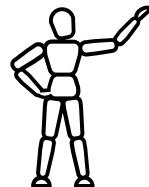
Bye

Call Stack

ssafy ()

Web API

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (15/19)

```
console.log('Hi')  
  
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)  
  
console.log('Bye')
```

Output

Hi

Bye

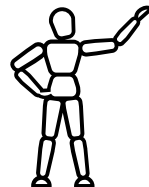
Call Stack

console.log('SSAFY')

ssafy ()

Web API

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (16/19)

```
console.log('Hi')  
  
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)  
  
console.log('Bye')
```

Output

Hi

Bye

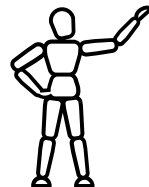
Call Stack

console.log('SSAFY')

ssafy ()

Web API

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (17/19)

```
console.log('Hi')
```

```
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)
```

```
console.log('Bye')
```

Output

Hi

Bye

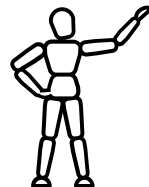
SSAFY

Call Stack

ssafy ()

Web API

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (18/19)

```
console.log('Hi')  
  
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)  
  
console.log('Bye')
```

```
console.log('Bye')
```

Output

Hi

Bye

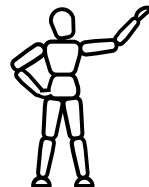
SSAFY

Call Stack

ssafy ()

Web API

Event Loop



Task Queue

JavaScript의 비동기 처리

그림으로 보는 비동기 처리(Runtime) (19/19)

종료 !

```
console.log('Hi')  
  
setTimeout(function ssafy() {  
  console.log('SSAFY')  
}, 3000)  
  
console.log('Bye')
```

Output

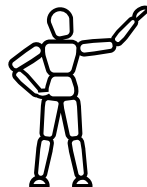
Hi
Bye
SSAFY

최종 출력

Call Stack

Web API

Event Loop



Task Queue

정리

- JavaScript는 한 번에 하나의 작업을 수행하는 Single Thread 언어로 동기적 처리를 하지만, 브라우저 환경에서는 Web API에서 처리된 작업이 지속적으로 Task Queue를 거쳐 Event Loop에 의해 Call Stack에 들어와 순차적으로 실행됨으로써 비동기 작업이 가능한 환경이 됨



이어서 ..

삼성 청년 SW 아카데미

Axios

Axios

- JavaScript의 HTTP 웹 통신을 위한 라이브러리
- 확장 가능한 인터페이스와 쉽게 사용할 수 있는 비동기 통신 기능을 제공
- node 환경은 npm을 이용해서 설치 후 사용할 수 있고,
browser 환경은 CDN을 이용해서 사용할 수 있음
- Axios 공식문서 및 github
 - <https://axios-http.com/kr/docs/intro>
 - <https://github.com/axios/axios>

Axios 기본 구조

Axios 사용해보기

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
  axios.get('요청할 URL')
    .then(성공하면 수행할 콜백함수)
    .catch(실패하면 수행할 콜백함수)
</script>
```

- get, post 등 여러 method 사용가능
- **then**을 이용해서 성공하면 수행할 로직을 작성
- **catch**를 이용해서 실패하면 수행할 로직을 작성

고양이 사진 가져오기

- The Cat API (<https://api.thecatapi.com/v1/images/search>)
 - 이미지를 요청해서 가져오는 작업을 비동기로 처리
- response 구조

```
// https://api.thecatapi.com/v1/images/search  
  
[  
  {  
    "id": "d6n",  
    "url": "https://cdn2.thecatapi.com/images/d6n.jpg",  
    "width": 333,  
    "height": 500  
  }  
]
```


고양이 사진 가져오기 (Python) (1/2)

- Python으로 요청해보기 (동기)

```
# 04_cat_api.py

import requests

print('고양이는 야옹')

cat_image_search_url = 'https://api.thecatapi.com/v1/images/search'
response = requests.get(cat_image_search_url)

if response.status_code == 200:
    print(response.json())
else:
    print('실패했다옹')

print('야옹야옹')
```



고양이 사진 가져오기 (Python) (2/2)

- Python으로 요청해보기 (동기)

```
# 04_cat_api.py

import requests

print('고양이는 야옹')

cat_image_search_url = 'https://api.thecatapi.com/v1/images/search'
response = requests.get(cat_image_search_url)

if response.status_code == 200:
    print(response.json())
else:
    print('실패했다옹')

print('야옹야옹')
```



```
고양이는 야옹
[{'id': 'cir', 'url': 'https://cdn2.thecatapi.com/images/cir.jpg', 'width': 500, 'height': 667}]
야옹야옹
```

고양이 사진 가져오기 (JavaScript) (1/2)

- Axios로 요청해보기 (비동기)

```
<!-- 05_cat_api.html -->  
  
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>  
<script>  
  console.log('고양이는 야옹')  
  const catImageSearchURL = 'https://api.thecatapi.com/v1/images/search'  
  
  axios.get(catImageSearchURL)  
    .then((response) => {  
      console.log(response.data)  
    })  
    .catch((error) => {  
      console.log('실패했다옹')  
    })  
    console.log('야옹야옹')  
</script>
```

JS

고양이 사진 가져오기 (JavaScript) (2/2)

- Axios로 요청해보기 (비동기)

```
<!-- 05_cat_api.html -->
```

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

```
<script>
```

```
  console.log('고양이는 야옹')
```

```
  const catImageSearchURL = 'https://api.thecatapi.com/v1/images/search'
```

```
  axios.get(catImageSearchURL)
```

```
    .then((response) => {
```

```
      console.log(response.data)
```

```
    })
```

```
    .catch((error) => {
```

```
      console.log('실패했다')
```

```
    })
```

```
    console.log('야옹야옹')
```

```
</script>
```

JS

고양이는 야옹

[aa.html:12](#)

야옹야옹

[aa.html:22](#)

▼ [{...}] ⓘ

[aa.html:17](#)

▶ 0: {id: '2so', url: 'https://cdn2.thecatapi.com/images/2so.jpg', length: 1}

▶ [[Prototype]]: Array(0)

고양이 사진 가져오기 (결과 비교)

- 동기식 코드(python)는 위에서부터 순서대로 처리가 되기때문에 첫번째 print 가 출력되고 이미지를 가져오는 처리를 기다렸다가 다음 print 가 출력되는 반면
- 비동기식 코드(JavaScript)는 바로 처리가 가능한 작업(console.log)은 바로 처리하고, 오래 걸리는 작업인 이미지를 요청하고 가져오는 일은 요청을 보내 놓고 기다리지 않고 다음 코드로 진행 후 완료가 된 시점에 결과 출력이 진행됨

| 고양이 사진 가져오기 (완성하기) (1/5)

- 작업 Flow

1. 버튼을 누르면
2. 고양이 이미지를 요청하고
3. 요청이 처리되어 응답이 오면
4. 처리된 response에 있는 url을 img태그에 넣어 보여주기

고양이 사진 가져오기 (완성하기) (2/5)

- 현재 HTML Body

```
<!-- 05_cat_api.html -->

<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
  console.log('고양이는 야옹')
  const catImageSearchURL = 'https://api.thecatapi.com/v1/images/search'

  axios.get(catImageSearchURL)
    .then((response) => {
      console.log(response.data)
    })
    .catch((error) => {
      console.log('실패했다옹')
    })
    console.log('야옹야옹')
</script>
```

고양이 사진 가져오기 (완성하기) (3/5)

(1) 버튼을 추가하고 이벤트 핸들러 부착

```
<button>야옹아 이리온</button>
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
  console.log('고양이는 야옹')
  const catImageSearchURL = 'https://api.thecatapi.com/v1/images/search'
  const btn = document.querySelector('button')

  btn.addEventListener('click', function () {
    axios.get(catImageSearchURL)
      .then((response) => {
        console.log(response.data)
      })
      .catch((error) => {
        console.log('실패했다옹')
      })
    console.log('야옹야옹')
  })
</script>
```


고양이 사진 가져오기 (완성하기) (4/5)

(2) 비동기 요청을 보내고, 응답이 오면 처리하기

```
<button>야옹아 이리온</button>
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
  console.log('고양이는 야옹')
  const catImageSearchURL = 'https://api.thecatapi.com/v1/images/search'
  const btn = document.querySelector('button')

  btn.addEventListener('click', function () {
    axios.get(catImageSearchURL)
      .then((response) => {
        imgElem = document.createElement('img')
        imgElem.setAttribute('src', response.data[0].url)
        document.body.appendChild(imgElem)
      })
      .catch((error) => {
        console.log('실패했다옹')
      })
      console.log('야옹야옹')
  })
</script>
```

고양이 사진 가져오기 (완성하기) (5/5)

(3) 완성 코드 (body)

```
<button>야옹아 이리온</button>
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
  console.log('고양이는 야옹')
  const catImageSearchURL = 'https://api.thecatapi.com/v1/images/search'
  const btn = document.querySelector('button')

  btn.addEventListener('click', function () {
    axios.get(catImageSearchURL)
      .then((response) => {
        imgElem = document.createElement('img')
        imgElem.setAttribute('src', response.data[0].url)
        document.body.appendChild(imgElem)
      })
      .catch((error) => {
        console.log('실패했다옹')
      })
      console.log('야옹야옹')
  })
</script>
```

고양이 사진 가져오기 (실행 결과)

- 버튼을 누르면 `console.log`가 먼저 출력되고 이미지 요청을 보낸다.
- 버튼을 여러 번 누르면 먼저 로딩되는 이미지부터 나오는 것을 볼 수 있다.



| 정리

- axios는 비동기로 데이터 통신을 가능하게 하는 라이브러리
- 같은 방식으로 우리가 배운 Django REST API로 요청을 보내서 데이터를 받아온 후 처리할 수 있음



이어서 ..

삼성 청년 SW 아카데미

Callback과 Promise

| 비동기 처리의 단점

- 비동기 처리의 핵심은 Web API로 들어오는 순서가 아니라
작업이 완료되는 순서에 따라 처리한다는 것!
- 그런데 이는 개발자 입장에서 코드의 실행 순서가 불명확하다는 단점이 있음
이와 같은 단점은 실행 결과를 예상하면서 코드를 작성할 수 없게 함
→ 어떻게 해야 할까? 🤔 → 콜백 함수를 사용하자 !

콜백 함수 (Callback Function)

콜백 함수(Callback Function)

| 콜백 함수란?

- 특별한 함수가 아님! 다른 함수의 인자로 전달되는 함수를 콜백 함수라고 한다.
- 비동기에만 사용되는 함수가 아니며 동기, 비동기 상관없이 사용 가능
- 시간이 걸리는 비동기 작업이 완료된 후 실행할 작업을 명시하는 데 사용되는 콜백 함수를 비동기 콜백(asynchronous callback)이라 부름

콜백 함수(Callback Function)

콜백 함수 예시

```
const btn = document.querySelector('button')
btn.addEventListener('click', () => {
  alert('Completed')
})
```

JavaScript의 Event Listener

```
from django.urls import path
from . import views

urlpatterns = [
    path('index/', views.index),
]
```

Django의 View Function

콜백 함수(Callback Function)

콜백 함수를 사용하는 이유

- 명시적인 호출이 아닌 특정한 조건 혹은 행동에 의해 호출되도록 작성할 수 있음
- “요청이 들어오면”, “이벤트가 발생하면”, “데이터를 받아오면” 등의 조건으로 이후 로직을 제어할 수 있음
- 비동기 처리를 순차적으로 동작할 수 있게 함
- 비동기 처리를 위해서는 콜백 함수의 형태가 반드시 필요함

콜백 함수(Callback Function)

콜백 지옥 (Callback Hell) (1/3)

- 콜백 함수는 연쇄적으로 발생하는 비동기 작업을 순차적으로 동작할 수 있게 함
- 보통 어떤 기능의 실행 결과를 받아서 다른 기능을 수행하기 위해 많이 사용하는데, 이 과정을 작성하다 보면 비슷한 패턴이 계속 발생하게 됨

A를 처리해서 결과가 나오면, 첫 번째 callback 함수를 실행하고
첫 번째 callback 함수가 종료되면, 두 번째 callback 함수를 실행하고
두 번째 callback 함수가 종료되면, 세 번째 callback 함수를 실행하고 ...



콜백 함수(Callback Function)

콜백 지옥 (Callback Hell) (2/3)

- 비동기 처리를 위한 콜백을 작성할 때 마주하는 문제를 Callback Hell(콜백 지옥)이라 하며, 그때의 코드 작성 형태가 마치 “피라미드와 같다”고 해서 “Pyramid of doom(파멸의 피라미드)”라고도 부름



콜백 함수(Callback Function)

콜백 지옥 (Callback Hell) (3/3)

```
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        }  
}
```



콜백 함수(Callback Function)

정리

- 콜백 함수는 비동기 작업을 순차적으로 실행할 수 있게 하는 반드시 필요한 로직
- 비동기 코드를 작성하다 보면 콜백 함수로 인한 콜백 지옥(callback hell)은 반드시 나타나는 문제
 - 코드의 가독성을 해치고
 - 유지 보수가 어려워짐



프로미스 (Promise)

프로미스(Promise)

- Callback Hell 문제를 해결하기 위해 등장한 비동기 처리를 위한 객체
- “작업이 끝나면 실행 시켜줄게”라는 약속(promise)
- 비동기 작업의 완료 또는 실패를 나타내는 객체
- Promise 기반의 클라이언트가 바로 이전에 사용한 **Axios** 라이브러리 !
 - “Promise based HTTP client for the browser and node.js”
 - 성공에 대한 약속 **then()**
 - 실패에 대한 약속 **catch()**

| then & catch (1/2)

- **then(callback)**

- 요청한 작업이 성공하면 callback 실행
- callback은 이전 작업의 성공 결과를 인자로 전달 받음

- **catch(callback)**

- then()이 하나라도 실패하면 callback 실행
- callback은 이전 작업의 실패 객체를 인자로 전달 받음

Callback과 Promise

then & catch (2/2)

- then과 catch 모두 항상 promise 객체를 반환
즉, 계속해서 **chaining**을 할 수 있음
- axios로 처리한 비동기 로직이 항상 promise 객체를 반환
그래서 then을 계속 이어 나가면서 작성할 수 있던 것

```
axios.get('요청할 URL').then(...).then(...).catch(...)
```

```
axios.get('요청할 URL') // Promise 객체 return
  .then(성공하면 수행할 1번 콜백함수)
  .then(1번 콜백함수가 성공하면 수행할 2번 콜백함수)
  .then(2번 콜백함수가 성공하면 수행할 3번 콜백함수)
  ...
  .catch(실패하면 수행할 콜백함수)
```

비동기 콜백 vs Promise (1/2)

// 기존의 콜백 함수 작성 방식

```
work1(function () {  
  // 첫번째 작업 ...  
  work2(result1, function (result2) {  
    // 두번째 작업 ...  
    work3(result2, function (result3) {  
      console.log('최종 결과 :' + result3)  
    })  
  })  
})
```

// promise 방식

```
work1()  
  .then((result1) => {  
    // work2  
    return result2  
  })  
  .then((result2) => {  
    // work3  
    return result3  
  })  
  .catch((error) => {  
    // error handling  
  })
```

| 비동기 콜백 vs Promise (2/2)

- promise 방식은 비동기 처리를
마치 우리가 일반적으로 위에서 아래로 적는
방식처럼 코드를 작성할 수 있음

```
// promise 방식

work1()
  .then((result1) => {
    // work2
    return result2
  })
  .then((result2) => {
    // work3
    return result3
  })
  .catch((error) => {
    // error handling
  })
```

Promise가 보장하는 것 (vs 비동기 콜백)

- 비동기 콜백 작성 스타일과 달리 Promise가 보장하는 특징
 1. callback 함수는 JavaScript의 Event Loop가
현재 실행 중인 Call Stack을 완료하기 이전에는 절대 호출되지 않음
 - Promise callback 함수는 Event Queue에 배치되는 엄격한 순서로 호출됨
 2. 비동기 작업이 성공하거나 실패한 뒤에 .then() 메서드를 이용하여 추가한 경우에도
1번과 똑같이 동작
 3. .then()을 여러 번 사용하여 여러 개의 callback 함수를 추가할 수 있음 (Chaining)
 - 각각의 callback은 주어진 순서대로 하나하나 실행하게 됨
 - Chaining은 Promise의 가장 뛰어난 장점

이어서 ..

삼성 청년 SW 아카데미

AJAX

AJAX란?

- Asynchronous JavaScript And XML (비동기식 JavaScript와 XML)
- 비동기 통신을 이용하면 화면 전체를 새로고침 하지 않아도 서버로 요청을 보내고, 데이터를 받아 화면의 일부분만 업데이트 가능
- 이러한 '비동기 통신 웹 개발 기술'을 AJAX라 함
- 비동기 웹 통신을 위한 라이브러리 중 하나가 Axios

AJAX 특징

- 페이지 전체를 reload(새로 고침)를 하지 않고서도 수행되는 “비동기성”
 - 서버의 응답에 따라 전체 페이지가 아닌 일부분만을 업데이트 할 수 있음
1. 페이지 새로고침 없이 서버에 요청
 2. 서버로부터 응답(데이터)을 받아 작업을 수행

비동기 적용하기

비동기(Async) 적용하기

| 사전 준비

- 마지막 Django 프로젝트 준비하기 (M:N까지 진행한 프로젝트)
- 가상 환경 생성 및 활성화, 패키지 설치

비동기(Async) 적용하기

| 팔로우 (follow)

- 각각의 템플릿에서 script 코드를 작성하기 위한 block tag 영역 작성

```
<!-- base.html -->

<body>
    ...
    {% block script %}
    {% endblock script %}
</body>
</html>
```

비동기(Async) 적용하기

| 팔로우 (follow)

- axios CDN 작성

```
<!-- accounts/profile.html -->
```

```
{% block script %}
```

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

```
<script>
```

```
</script>
```

```
{% endblock script %}
```

| 팔로우 (follow)

- form 요소 선택을 위해 id 속성 지정 및 선택
- 불필요해진 action과 method 속성은 삭제 (요청은 axios로 대체되기 때문)

```
<!-- accounts/profile.html -->  
  
<form id="follow-form">  
  ...  
</form>
```

```
<!-- accounts/profile.html -->  
  
<script>  
  const form = document.querySelector('#follow-form')  
</script>
```

| 팔로우 (follow)

- form 요소에 이벤트 핸들러 작성 및 submit 이벤트 취소

```
<!-- accounts/profile.html -->

<script>
  const form = document.querySelector('#follow-form')
  form.addEventListener('submit', function (event) {
    event.preventDefault()
  })
</script>
```


비동기(Async) 적용하기

| 팔로우 (follow)

- axios 요청 준비

```
<!-- accounts/profile.html -->

<script>
  const form = document.querySelector('#follow-form')
  form.addEventListener('submit', function (event) {
    event.preventDefault()
    axios({
      method: 'post ',
      url: `/accounts/${{???}}/follow/`,
    })
  })
</script>
```

| 팔로우 (follow)

- 현재 axios로 POST 요청을 보내기 위해 필요한 것
 1. url에 작성할 user pk는 어떻게 작성해야 할까?
 2. csrftoken은 어떻게 보내야 할까?

| 팔로우 (follow)

- 현재 axios로 POST 요청을 보내기 위해 필요한 것
 1. url에 작성할 user pk는 어떻게 작성해야 할까?
 2. csrftoken은 어떻게 보내야 할까?

비동기(Async) 적용하기

| 팔로우 (follow)

- url에 작성할 user pk 가져오기 (HTML -> JavaScript)

```
<!-- accounts/profile.html -->  
  
<form id="follow-form" data-user-id="{{ person.pk }}">  
  ...  
</form>
```

```
<!-- accounts/profile.html -->  
  
<script>  
  const form = document.querySelector('#follow-form')  
  form.addEventListener('submit', function (event) {  
    event.preventDefault()  
  
    const userId = event.target.dataset.userId  
    ...  
  })  
</script>
```

비동기(Async) 적용하기

| data-* attributes (1/2)

- 사용자 지정 데이터 특성을 만들어 임의의 데이터를 HTML과 DOM사이에서 교환 할 수 있는 방법
- 사용 예시

```
<div data-my-id="my-data"></div>  
<script>  
  const myId = event.target.dataset.myId  
</script>
```

- 모든 사용자 지정 데이터는 dataset 속성을 통해 사용할 수 있음

https://developer.mozilla.org/ko/docs/Web/HTML/Global_attributes/data-*

| data-* attributes (2/2)

- 예를 들어 **data-test-value** 라는 이름의 특성을 지정했다면
JavaScript에서는 **element.dataset.testValue**로 접근할 수 있음
- 속성명 작성 시 주의사항
 - 대소문자 여부에 상관없이 xml로 시작하면 안 됨
 - 세미콜론을 포함해서는 안됨
 - 대문자를 포함해서는 안됨

비동기(Async) 적용하기

| 팔로우 (follow)

- url 작성 마치기

```
<!-- accounts/profile.html -->

<script>
  const form = document.querySelector('#follow-form')
  form.addEventListener('submit', function (event) {
    event.preventDefault()
    const userId = event.target.dataset.userId
    axios({
      method: 'post',
      url: `/accounts/${userId}/follow/`,
    })
  })
</script>
```

| 팔로우 (follow)

- 현재 axios로 POST 요청을 보내기 위해 필요한 것
 1. url에 작성할 user pk는 어떻게 작성해야 할까?
 2. csrftoken은 어떻게 보내야 할까?

비동기(Async) 적용하기

팔로우 (follow)

- 먼저 hidden 타입으로 숨겨져있는 csrf 값을 가진 input 태그를 선택해야 함

<https://docs.djangoproject.com/en/3.2/ref/csrf/>

The screenshot displays a web application interface on the left and its corresponding HTML structure in the browser's developer tools on the right.

Web Interface (Left):

- Links: [Login](#) [Signup](#)
- Section: **test1님의 프로필**
- Text: 팔로워 : 1 / 팔로잉 : 0
- Button: 팔로우
- Section: **test1이 작성한 모든 게시글**
- Section: **test1이 작성한 모든 댓글**
- Section: **test1이 좋아요 한 모**

Developer Tools (Right):

- Tab: Elements
- HTML Structure:

```
<!-- base.html -->
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div class="container">
      <a href="/accounts/login/">Login</a>
      <a href="/accounts/signup/">Signup</a>
      <hr>
      <h1>test1님의 프로필</h1>
      <div>...</div>
      <div>
        <form id="follow-form" data-user-id="1">
          <input type="hidden" name="csrfmiddlewaretoken" value="PwqWK20x1M8uRNUZ1omCjeDpYcgRY54YvJrE0ipkoc6mn7tgqRj0reEKFs2FkHk6"> == $0
          <input type="submit" value="팔로우">
        </form>
      </div>
    </div>
  </body>
</html>
```
- A red box highlights the hidden input field: `<input type="hidden" name="csrfmiddlewaretoken" value="PwqWK20x1M8uRNUZ1omCjeDpYcgRY54YvJrE0ipkoc6mn7tgqRj0reEKFs2FkHk6"> == $0`

비동기(Async) 적용하기

| 팔로우 (follow)

- 먼저 hidden 타입으로 숨겨져있는 csrf 값을 가진 input 태그를 선택해야 함

<https://docs.djangoproject.com/en/3.2/ref/csrf/>

```
<!-- accounts/profile.html -->

<script>
  const form = document.querySelector('#follow-form ' )
  const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value
</script>
```

비동기(Async) 적용하기

팔로우 (follow)

- AJAX로 csrftoken을 보내는 방법

<https://docs.djangoproject.com/en/3.2/ref/csrf/#setting-the-token-on-the-ajax-request>

```
<!-- accounts/profile.html -->

<script>
  const form = document.querySelector('#follow-form ' )
  const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value
  form.addEventListener('submit', function (event) {
    event.preventDefault()
    const userId = event.target.dataset.userId
    axios({
      method: 'post ' ,
      url: `/accounts/${userId}/follow/`,
      headers: {'X-CSRFToken': csrftoken,}
    })
  })
</script>
```

| 팔로우 (follow)

- 팔로우 버튼을 토글하기 위해서는 현재 팔로우가 된 상태인지 여부 확인이 필요
- axios 요청을 통해 받는 response 객체를 활용해 view 함수를 통해서 팔로우 여부를 파악 할 수 있는 변수를 담아 JSON 타입으로 응답하기

비동기(Async) 적용하기

팔로우 (follow)

- 팔로우 여부를 확인하기 위한 is_followed 변수 작성 및 JSON 응답

```
# accounts/views.py

from django.http import JsonResponse

@require_POST
def follow(request, user_pk):
    if request.user.is_authenticated:
        User = get_user_model()
        me = request.user
        you = User.objects.get(pk=user_pk)
        if me != you:
            if you.followers.filter(pk=me.pk).exists():
                you.followers.remove(me)
                is_followed = False
            else:
                you.followers.add(me)
                is_followed = True
            context = {
                'is_followed': is_followed,
            }
            return JsonResponse(context)
        return redirect('accounts:profile', you.username)
    return redirect('accounts:login')
```

팔로우 (follow)

- view 함수에서 응답한 is_followed를 사용해 버튼 토글하기

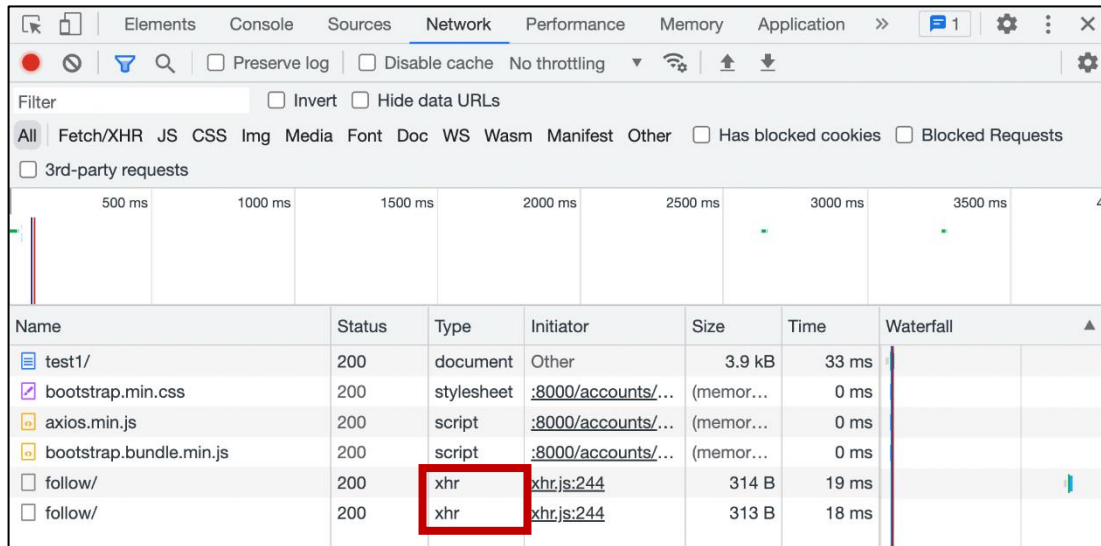
```
<!-- accounts/profile.html -->

<script>
  ...
  axios({
    method: 'post',
    url: `/accounts/${userId}/follow/`,
    headers: {'X-CSRFToken': csrftoken},
  })
  .then((response) => {
    const isFollowed = response.data.is_followed
    const followBtn = document.querySelector('#follow-form > input[type=submit]')
    if (isFollowed === true) {
      followBtn.value = '언팔로우'
    } else {
      followBtn.value = '팔로우'
    }
  })
</script>
```

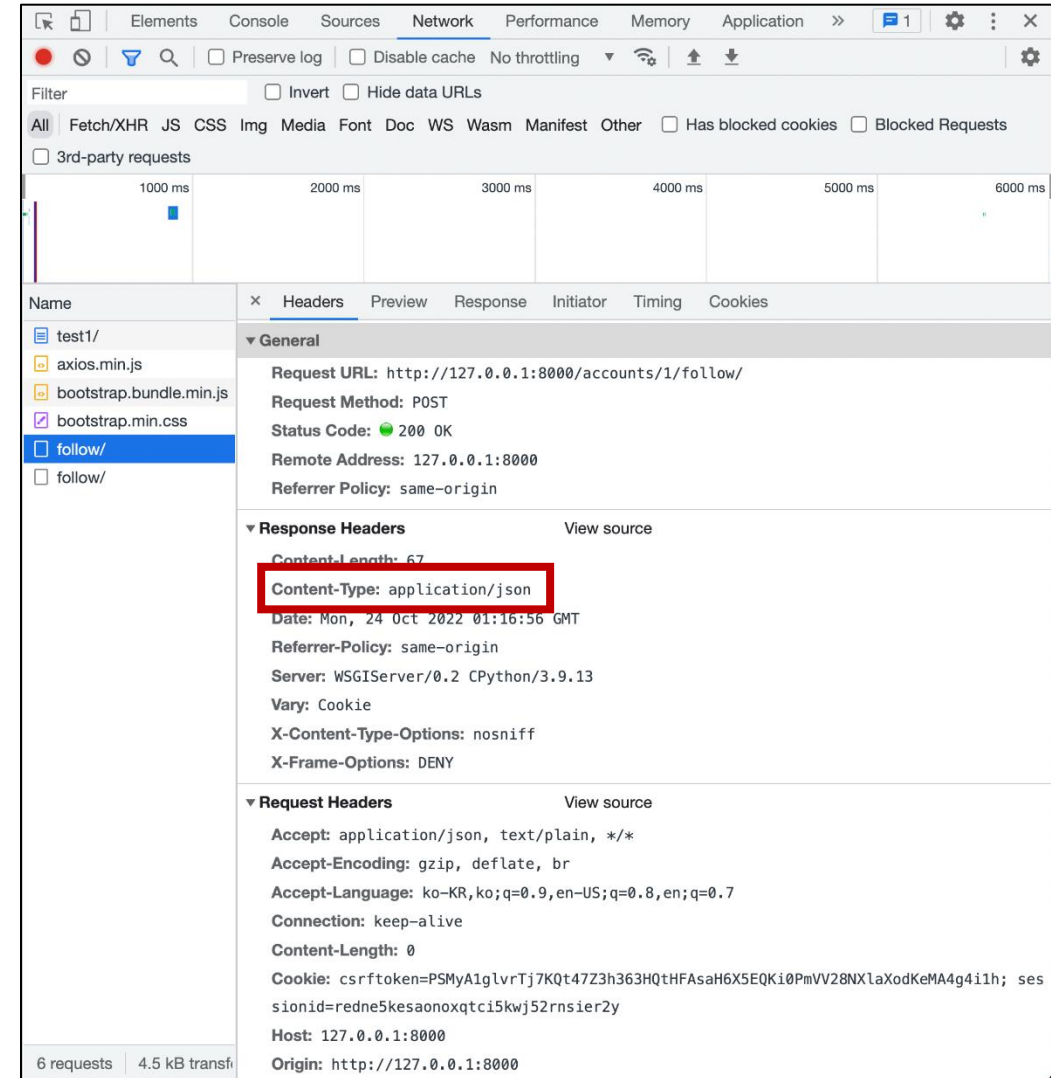
비동기(Async) 적용하기

팔로우 (follow)

- 결과 확인 (개발자 도구 - Network)



Name	Status	Type	Initiator	Size	Time	Waterfall
test1/	200	document	Other	3.9 kB	33 ms	
bootstrap.min.css	200	stylesheet	:8000/accounts/...	(memor...)	0 ms	
axios.min.js	200	script	:8000/accounts/...	(memor...)	0 ms	
bootstrap.bundle.min.js	200	script	:8000/accounts/...	(memor...)	0 ms	
follow/	200	xhr	xhr.js:244	314 B	19 ms	
follow/	200	xhr	xhr.js:244	313 B	18 ms	



Network tab details for the 'follow/' request:

- General:**
 - Request URL: http://127.0.0.1:8000/accounts/1/follow/
 - Request Method: POST
 - Status Code: 200 OK
 - Remote Address: 127.0.0.1:8000
 - Referrer Policy: same-origin
- Response Headers:**
 - Content-Length: 67
 - Content-Type: application/json**
 - Date: Mon, 24 Oct 2022 01:16:56 GMT
 - Referrer-Policy: same-origin
 - Server: WSGIServer/0.2 CPython/3.9.13
 - Vary: Cookie
 - X-Content-Type-Options: nosniff
 - X-Frame-Options: DENY
- Request Headers:**
 - Accept: application/json, text/plain, */*
 - Accept-Encoding: gzip, deflate, br
 - Accept-Language: ko-KR, ko;q=0.9,en-US;q=0.8,en;q=0.7
 - Connection: keep-alive
 - Content-Length: 0
 - Cookie: csrftoken=PSMyA1glvrTj7KQt47Z3h363HQthFAsaH6X5EQKi0PmVV28NX1aXodKeMA4gi1h; sessionid=redne5kesaonoxqtc15kwj52rnsier2y
 - Host: 127.0.0.1:8000
 - Origin: http://127.0.0.1:8000

[참고] XHR

- "XMLHttpRequest"
- AJAX 요청을 생성하는 JavaScript API
- XHR의 메서드로 브라우저와 서버 간 네트워크 요청을 전송할 수 있음
- Axios는 손쉽게 XHR을 보내고 응답 결과를 Promise 객체로 반환해주는 라이브러리

팔로워 & 팔로잉 수 비동기 적용 (1/4)

- 해당 요소를 선택할 수 있도록 span 태그와 id 속성 작성

```
<!-- accounts/profile.html -->

{% extends 'base.html' %}

{% block content %}
  <h1>{{ person.username }}님의 프로필</h1>
  <div>
    팔로워 : <span id="followers-count">{{ person.followers.all|length }}</span> /
    팔로잉 : <span id="followings-count">{{ person.followings.all|length }}</span>
  </div>
```

팔로워 & 팔로잉 수 비동기 적용 (2/4)

- 직전에 작성한 span 태그를 각각 선택

```
<!-- accounts/profile.html -->

<script>
  ...
  axios({
    method: 'post',
    url: `/accounts/${userId}/follow/`,
    headers: {'X-CSRFToken': csrftoken,}
  })
  .then((response) => {
    ...
    const followersCountTag = document.querySelector('#followers-count')
    const followingsCountTag = document.querySelector('#followings-count')
  })
</script>
```

비동기(Async) 적용하기

팔로워 & 팔로잉 수 비동기 적용 (3/4)

- 팔로워, 팔로잉 인원 수 연산은 view 함수에서 진행하여 결과를 응답으로 전달

```
# accounts/views.py

@require_POST
def follow(request, user_pk):
    ...
    context = {
        'is_followed': is_followed,
        'followers_count': you.followers.count(),
        'followings_count': you.followings.count(),
    }
    return JsonResponse(context)
    return redirect('accounts:profile', you.username)
    return redirect('accounts:login')
```

팔로워 & 팔로잉 수 비동기 적용 (4/4)

- view 함수에서 응답한 연산 결과를 사용해 각 태그의 인원 수 값 변경하기

```
<!-- accounts/profile.html -->

<script>
  ...
  axios({
    method: 'post',
    url: `/accounts/${userId}/follow/`,
    headers: {'X-CSRFToken': csrftoken,}
  })
  .then((response) => {
    ...
    const followersCount = response.data.followers_count
    const followingsCount = response.data.followings_count
    followersCountTag.innerText = followersCount
    followingsCountTag.innerText = followingsCount
  })
</script>
```

비동기(Async) 적용하기

최종 코드 (1/3)

- HTML 코드

```
<!-- accounts/profile.html -->

{% extends 'base.html' %}

{% block content %}
  <h1>{{ person.username }}님의 프로필</h1>
  <div>
    팔로워 : <span id="followers-count">{{ person.followers.all|length }}</span> /
    팔로잉 : <span id="followings-count">{{ person.followings.all|length }}</span>
  </div>

  {% if request.user != person %}
  <div>
    <form id="follow-form" data-user-id="{{ person.pk }}">
      {% csrf_token %}
      {% if request.user in person.followers.all %}
        <input type="submit" value="언팔로우">
      {% else %}
        <input type="submit" value="팔로우">
      {% endif %}
    </form>
  </div>
  {% endif %}
...

```

비동기(Async) 적용하기

최종 코드 (2/3)

- Python 코드

```
# accounts/views.py

@require_POST
def follow(request, user_pk):
    if request.user.is_authenticated:
        User = get_user_model()
        me = request.user
        you = User.objects.get(pk=user_pk)
        if me != you:
            if you.followers.filter(pk=me.pk).exists():
                you.followers.remove(me)
                is_followed = False
            else:
                you.followers.add(me)
                is_followed = True
        context = {
            'is_followed': is_followed,
            'followers_count': you.followers.count(),
            'followings_count': you.followings.count(),
        }
        return JsonResponse(context)
    return redirect('accounts:profile', you.username)
return redirect('accounts:login')
```

비동기(Async) 적용하기

최종 코드 (3/3)

- JavaScript 코드

```
<!-- accounts/profile.html -->
const form = document.querySelector('#follow-form')
const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

form.addEventListener('submit', function (event) {
  event.preventDefault()
  const userId = event.target.dataset.userId

  axios({
    method: 'post ',
    url: `/accounts/${userId}/follow/`,
    headers: {'X-CSRFToken': csrftoken,}
  })
  .then((response) => {
    const isFollowed = response.data.is_followed
    const followBtn = document.querySelector('#follow-form > input[type=submit] ')
    if (isFollowed === true) {
      followBtn.value = '언팔로우 '
    } else {
      followBtn.value = '팔로우 '
    }
    const followersCountTag = document.querySelector('#followers-count ')
    const followingsCountTag = document.querySelector('#followings-count ')
    const followersCount = response.data.followers_count
    const followingsCount = response.data.followings_count
    followersCountTag.innerText = followersCount
    followingsCountTag.innerText = followingsCount
  })
  .catch((error) => {
    console.log(error.response)
  })
})
```

비동기(Async) 적용하기

| 좋아요 (like)

- 좋아요 비동기 적용은 “팔로우와 동일한 흐름 + `forEach()` & `querySelectorAll()`”
 - index 페이지 각 게시글에 좋아요 버튼이 있기 때문

비동기(Async) 적용하기

최종 코드 (1/3)

- HTML 코드

```
<!-- articles/index.html -->

{% extends 'base.html' %}

{% block content %}
<h1>Articles</h1>
{% if request.user.is_authenticated %}
  <a href="{% url 'articles:create' %}">CREATE</a>
{% endif %}
<hr>
{% for article in articles %}
<p>
  <b>작성자 : <a href="{% url 'accounts:profile' article.user %}">{{ article.user }}</a></b>
</p>
<p>글 번호 : {{ article.pk }}</p>
<p>제목 : {{ article.title }}</p>
<p>내용 : {{ article.content }}</p>
<div>
  <form class="like-forms" data-article-id="{{ article.pk }}">
    {% csrf_token %}
    {% if request.user in article.like_users.all %}
      <input type="submit" value="좋아요 취소" id="like-{{ article.pk }}">
    {% else %}
      <input type="submit" value="좋아요" id="like-{{ article.pk }}">
    {% endif %}
  </form>
</div>
<a href="{% url 'articles:detail' article.pk %}">상세 페이지</a>
<hr>
{% endfor %}
{% endblock content %}
```

비동기(Async) 적용하기

최종 코드 (2/3)

- Python 코드

```
# articles/views.py

from django.http import JsonResponse

@require_POST
def likes(request, article_pk):
    if request.user.is_authenticated:
        article = Article.objects.get(pk=article_pk)

        if article.like_users.filter(pk=request.user.pk).exists():
            article.like_users.remove(request.user)
            is_liked = False
        else:
            article.like_users.add(request.user)
            is_liked = True
        context = {
            'is_liked': is_liked,
        }
        return JsonResponse(context)
    return redirect('accounts:login')
```

비동기(Async) 적용하기

최종 코드 (3/3)

- JavaScript 코드

```
<!-- articles/index.html -->

const forms = document.querySelectorAll('.like-forms')
const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

forms.forEach((form) => {
  form.addEventListener('submit', function (event) {
    event.preventDefault()
    const articleId = event.target.dataset.articleId

    axios({
      method: 'post',
      url: `http://127.0.0.1:8000/articles/${articleId}/likes/`,
      headers: {'X-CSRFToken': csrftoken},
    })
    .then((response) => {
      const isLiked = response.data.is_liked
      const likeBtn = document.querySelector(`#like-${articleId}`)
      if (isLiked === true) {
        likeBtn.value = '좋아요 취소'
      } else {
        likeBtn.value = '좋아요'
      }
    })
    .catch((error) => {
      console.log(error.response)
    })
  })
})
```

이어서 ..

삼성 청년 SW 아카데미

마무리

왜 비동기(Asynchronous) 방식이 필요할까

- “human-centered design with UX”
 - “인간 중심으로 설계된 사용자 경험”
 - 실제 Ajax라는 용어를 처음 논문에서 사용한 Jesse James Garrett이 Ajax를 소개하며 강조한 한 마디



마무리

- 동기와 비동기
- JavaScript의 비동기 처리
 - Call Stack, Web API, Task Queue, Event Loop
- Axios 라이브러리
 - then & catch
- Async Callback과 Promise
- AJAX

다음 방송에서 만나요!

삼성 청년 SW 아카데미