JavaScript #3

HTTP와 fetch api

DEVKOR

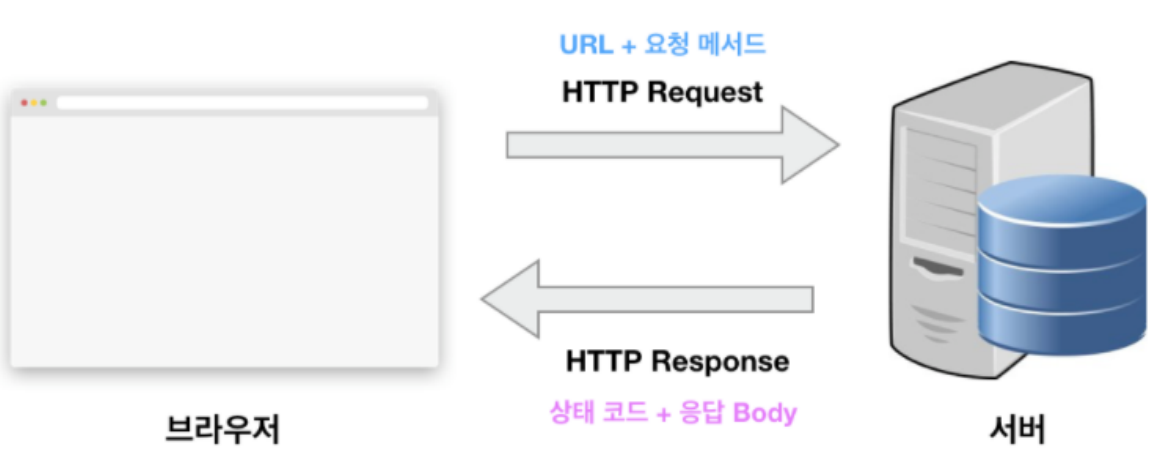
http

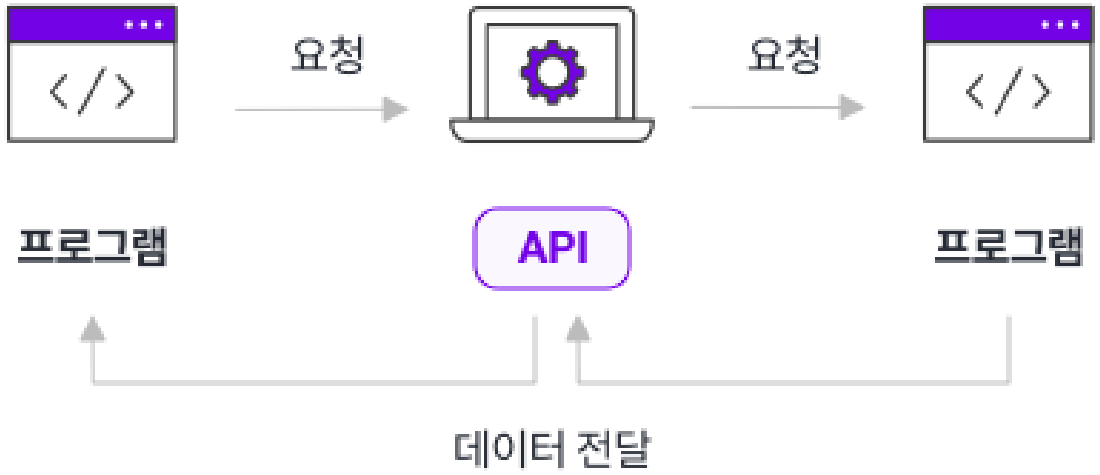
fetch api

callback & asynchronicity

promise

async & await





http://blog.wishket.com/api%EB%9E%80-%EC%89%BD%EA%B2%8C-%EC%84%A4%EB%AA%85-%EA%B7%B8%EB%A6%B0%ED%81%B4%EB%9D%BC%EC%9D%B4%EC%96%B8%ED%8A%B8/

DEVKOR

Fetch API

The Fetch API provides an interface for fetching resources (including across the network). It will seem familiar to anyone who has used `XMLHttpRequest`, but the new API provides a more powerful and flexible feature set.

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

```
1 const fetchResponsePromise = fetch(resource [, init])
```

```
1 fetch(url, {
2   method: "POST",
3   mode: 'cors',
4   cache: 'no-cache',
5   headers:  {"Content-Type", "application/json"},
6   credentials: "same-origin",
7   body: JSON.stringify(bodyData)
8 });
```

callback : 임의의 함수에서 인자로 받아들인 함수를 호출하는 것

```
1 function f1() {
2   console.log(1);
3 }
4
5 function f2(f) {
6   console.log("Run f1");
7   f();
8 }
```

```
1 function f0() {
2   console.log(0);
3 }
4 function f1() {
5   console.log(1);
6 }
7
8 // function setInterval(handler: TimerHandler, timeout?: number)
9 setInterval(f0, 1000);
10
11 // function setTimeout(handler: TimerHandler, timeout?: number)
12 setTimeout(f1, 10000);
```

DEVKOR

asynchronicity

```
function fStart() {
  console.log('start');
}

function fEnd() {
  console.log('end');
}

function fDone() {
  console.log('done');
}

fStart();
setTimeout(fEnd, 5000);
fDone();
```

```
let url = "https://jsonplaceholder.typicode.com/users/1";

function getData1() {
  let result = 0;
  $.get(url, function (response) {
    result = response;
  });
  return result;
}

console.log("getData1: ", getData1());
```

DEVKOR

synchronicity

```
let url = "https://jsonplaceholder.typicode.com/users/1";

function getData2(callbackFunc) {
  $.get(url, function (response) {
    callbackFunc(response);
  });
}

getData2(console.log);
```

callbacks hell -> heavy code

```
1 $.get(url, function (response) {
2   parseValue(response, function (id) {
3     auth(id, function (result) {
4       display(result, function (text) {
5         console.log(text);
6       });
7     });
8   });
9 });
```

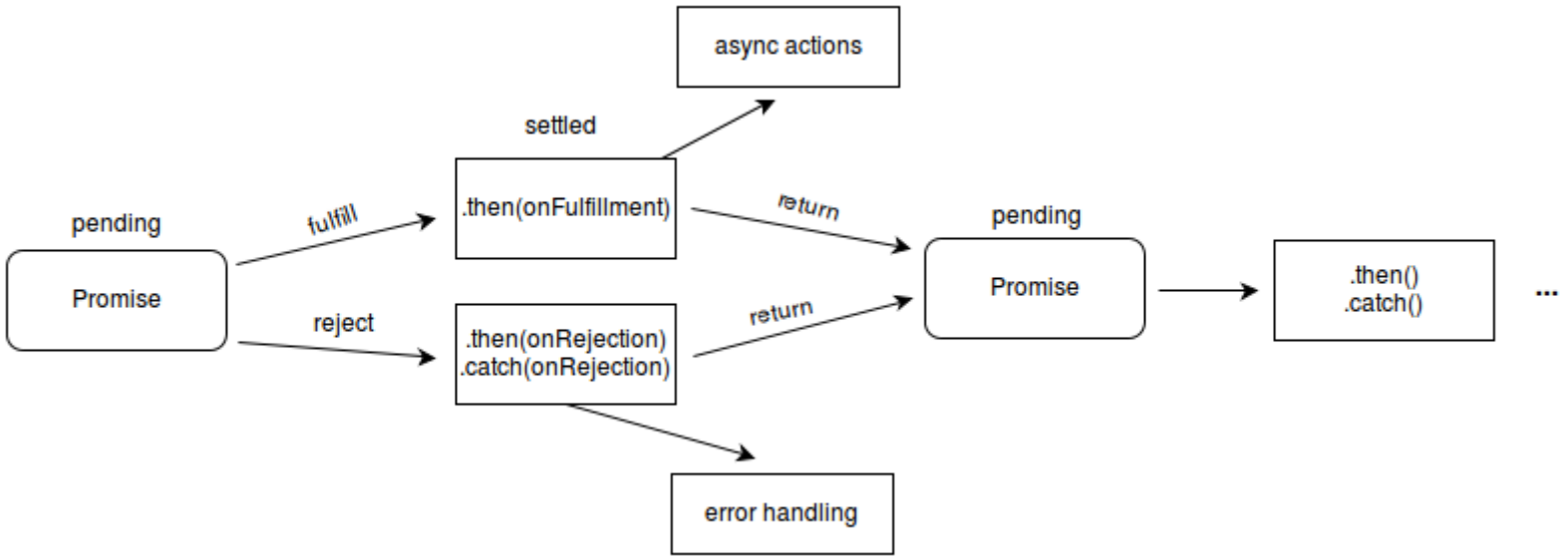
callbacks hell -> heavy code

```
1 $.get(url, function (response) {
2   parseValue(response, function (id) {
3     auth(id, function (result) {
4       display(result, function (text) {
5         console.log(text);
6       });
7     });
8   });
9 });
```

promise

async & await

`Promise` 객체는 비동기 작업이 맞이할 미래의 완료 또는 실패와 그 결과 값을 나타냅니다.



https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

How to handle promise?

```
let url = "https://jsonplaceholder.typicode.com/users/1";

const fetchPromise = fetch(url);
console.log(fetchPromise);
```

```
fetchPromise
  .then(res => {
    console.log(res);
  });
```

```
function callback1(res) {
  console.log(res);
};

fetchPromise
  .then(callback1);
```

```
fetchPromise
  .then((res) => {
    return res.json();
  })
  .then((data) => {
    console.log("data: ", data);
    return data.name
  })
  .then((name) => {
    console.log("name: ", name);
  });
```

DEVKOR

Error Handling

```
var myPromise = function (param) {
  return new Promise(function (resolve, reject) {
    setTimeout(function () {
      if (param) {
        resolve(1);
      }
      else {
        reject(-1);
      }
    }, 5000);
  });
};
```

```
function resolveFunc(res) {
  console.log("resolved: ", res);
};

function rejectFunc(err) {
  console.log("rejected: ", err);
};
```

```
myPromise(false)
  .then(resolveFunc)
  .catch(rejectFunc);
```

```
myPromise(false)
  .then(resolveFunc, rejectFunc)
```

DEVKOR

예제

promise

async & await

async & await

✓await 연산자는 Promise를 기다리기 위해 사용됩니다.

✓연산자는 async function 내부에서만 사용할 수 있습니다.

✓await 문은 Promise가 fulfill되거나 reject 될 때까지 async 함수의 실행을 일시 정지하고,

✓Promise가 fulfill되면 async 함수를 일시 정지한 부분부터 실행합니다.

✓이때 await 문의 반환값은 Promise 에서 fulfill된 값이 됩니다.

✓만약 Promise가 reject되면, await 문은 reject된 값을 throw합니다.

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/await

async & await

```
 1 function resolveAfter2Seconds(x) {
 2   return new Promise(resolve => {
 3     setTimeout(() => {
 4       resolve(x);
 5     }, 2000);
 6   });
 7 }
 8
 9 async function f1() {
10   var x = await resolveAfter2Seconds(10);
11   console.log(x); // 10
12 }
13
14 f1();
```

✓ await 연산자는 Promise를 기다리기 위해 사용됩니다.

✓ 연산자는 async function 내부에서만 사용할 수 있습니다.

✓ await 문은 Promise가 fulfill되거나 reject 될 때까지 async 함수의 실행을 일시 정지하고,

✓ Promise가 fulfill되면 async 함수를 일시 정지한 부분부터 실행합니다.

✓ 이때 await 문의 반환값은 Promise 에서 fulfill된 값이 됩니다.

async & await: error handling

```
async function f3() {
  try {
    var z = await Promise.reject(30);
  } catch(e) {
    console.log(e); // 30
  }
}
f3();
```

✓만약 Promise가 reject되면, await 문은 reject된 값을 throw합니다.

```
var response = await promisedFunction()
    .catch((err) => {
      console.error(err);
    });
// response will be undefined if the promise is rejected
```

DEVKOR

async & await

```
async function getName1() {
  let name = await fetch(url)
    .then(getResponse)
    .then(parseName)
    .catch((err) => {
      console.error(err);
    });
  console.log("getName: ", name);
}
```

```
async function getName2() {
  let response = await fetch(url);
  let data = await getResponse(response);
  let name = await parseName(data);
  console.log("getName: ", name);
}
```

DEVKOR