JavaScript 101 - (11) 비동기 통신과 Promise

자바스크립트에서 비동기 통신을 위해 사용되는 방법으로는 callback, promise, async-await이 있습니다. callback은 비동기 처리가 완료된 후 실행될 함수를 인자로 전달하는 방식입니다. promise는 비동기 처리가 완료되면 결과를 반환하는 객체입니다. async-await은 promise를 기반으로 하며, 비동기 처리가 완료될 때까지 기다린 후 결과를 반환합니다.

callback은 비동기 처리가 복잡해질수록 콜백 지옥에 빠질 가능성이 있습니다. promise는 콜백 지옥을 해결하고, 비동기 처리를 순차적으로 처리할 수 있습니다. async-await은 promise를 더욱 간결하게 사용할 수 있도록 해주며, 비동기 처리를 동기적으로 처리할 수 있습니다.

비동기 콜백 함수

비동기(Asynchronous) 함수란 쉽게 설명하면 호출부에서 실행 결과를 기다리지 않아도 되는 함수입니다. 반대로 동기(Synchronous) 함수는 호출부에서 실행 결과가 리턴될 때까지 기다려야 하는 함수입니다.

콜백함수(Callback Function)은 비동기 처리를 위해 사용될 수 있습니다. 콜백함수란 파라미터로 함수를 전달받아, 함수의 내부에서 실행하는 함수입니다.

```
function asyncFunction(callback) {
    setTimeout(function() {
        callback('Async Hello world');
    }, 16);
}

asyncFunction(function(data) {
    console.log(data);
});
```

이 예제에서 `asyncFunction`은 비동기 함수로, `setTimeout`을 사용하여 16ms 후에 콜백함수를 실행합니다. 이 콜백함수는 `asyncFunction`의 인자로 전달되며, `Async Hello world` 문자열을 출력합니다.

콜백지옥 (Callback Hell)

콜백 지옥(callback hell)은 콜백 함수를 익명 함수로 전달하는 과정에서 또 다시 콜백 안에 함수 호출이 반복되어 코드의 들여쓰기 순준이 감당하기 힘들 정도로 깊어지는 현상을 말합니다. 이러한 콜백 지옥은 비동기 처리가 복잡해질수록 발생할 가능성이 높습니다.

- 콜백헬 예시

```
getImage('./image.png', (image, err) => {
  if (err) throw new Error(err)
  compressImage(image, (compressedImage, err) => {
    if (err) throw new Error(err)
    applyFilter(compressedImage, (filteredImage, err) => {
      if (err) throw new Error(err)
      saveImage(compressedImage, (res, err) => {
        if (err) throw new Error(err)
        console.log("Successfully saved image!")
      })
    })
  })
})
```

```
function taskA(a, b, cb) {
  setTimeout(() => {
    const res = a + b;
    cb(res);
  }, 2000);
}

function taskB(a, cb) {
  setTimeout(() => {
    const res = a * 2;
    cb(res);
  }, 2000);
}

function taskC(a, cb) {
  setTimeout(() => {
    const res = a * -1;
    cb(res);
  }, 2000);
}

taskA(1, 2, (res_a) => {
  taskB(res_a, (res_b) => {
    taskC(res_b, (res_c) => {
      console.log("taskC Result : ", res_c);
    });
  });
```

콜백 지옥을 해결하는 방법에는 Promise 나 Async를 사용하는 방법이 있습니다.

Promise

Promise는 자바스크립트에서 비동기 처리를 위한 객체입니다. Promise를 사용하면 비동기 처리 결과를 콜백함수 대신에 직접 다룰 수 있습니다. Promise 객체를 생성하고, 비동기 처리를 수행하는 함수를 만들고, Promise 객체의 `resolve`나 `reject` 함수를 호출하면 됩니다. Promise 객체의 `then` 메서드나 `catch` 메서드를 사용해서 비동기 처리 결과를 처리할 수 있습니다.

Promise는 비동기 작업의 결과를 나타내는 객체입니다. 이를 이용하여 비동기 작업을 순차적으로 처리하거나, 여러 개의 비동기 작업을 병렬로 처리할 수 있습니다. Promise는 다음과 같은 상태를 갖습니다.

- `Pending`: 비동기 작업이 아직 수행되지 않은 상태
- `Fulfilled`: 비동기 작업이 성공적으로 완료된 상태
- `Rejected`: 비동기 작업이 실패한 상태

Promise는 다음과 같은 메서드를 제공합니다.

- `then`: 비동기 작업이 성공적으로 완료된 경우 호출되는 메서드
- `catch`: 비동기 작업이 실패한 경우 호출되는 메서드
- `finally`: 비동기 작업의 결과에 상관없이 호출되는 메서드

예를 들어, Promise 객체를 생성하고, 비동기 처리를 수행하는 함수를 만들어서 Promise 객체의 `resolve` 함수를 호출하면, `then` 메서드를 사용해서 비동기 처리 결과를 처리할 수 있습니다.

```
const promise = new Promise((resolve, reject) => {
  // 비동기 처리를 수행하는 함수
  // 비동기 처리가 완료되면 resolve 함수를 호출합니다.
  resolve('비동기 처리 결과');
});

promise.then((result) => {
  // 비동기 처리 결과를 처리합니다.
  console.log(result);
}).catch((error) => {
  // 에러를 처리합니다.
  console.error(error);
});
```

Promise를 사용하는 방법

1. Promise 객체를 생성합니다.
2. 비동기 처리를 수행하는 함수를 만듭니다.

3. Promise 객체의 resolve나 reject 함수를 호출합니다.
4. Promise 객체의 then 메서드나 catch 메서드를 사용해서 비동기 처리 결과를 처리합니다.

Promise를 이용하여 비동기 작업을 처리하면 중첩된 코드가 줄어들어 가독성이 좋아지고 유지보수가 용이해집니다. 하지만 Promise를 이용하여 비동기 작업을 처리하는 경우, Promise 체인이 길어질 수 있어 코드의 가독성이 떨어지는 경우가 발생할 수 있습니다. 이를 해결하기 위해 ES8에서는 async/await 문법을 도입하였습니다.

async-await

async-await는 Promise를 더욱 쉽게 사용할 수 있도록 해주는 문법입니다. `async` 함수를 만들고, 함수 내부에서 비동기 처리를 수행하는 코드 앞에 `await` 키워드를 붙이면 됩니다. `await` 키워드를 붙인 코드는 Promise 객체를 반환합니다. `async` 함수에서 Promise 객체를 반환하면, `then` 메서드나 `catch` 메서드를 사용해서 비동기 처리 결과를 처리할 수 있습니다.

async/await는 Promise를 기반으로 한 문법입니다. 이를 이용하여 동기적인 코드처럼 비동기 작업을 처리할 수 있습니다. async/await 문법을 이용하면 Promise를 이용하여 작성한 코드보다 가독성이 더욱 좋아집니다.

예를 들어, async 함수를 만들고, 함수 내부에서 Promise 객체를 반환하는 코드 앞에 `await` 키워드를 붙이면, 비동기 처리 결과를 직접 다룰 수 있습니다.

```
async function asyncFunction() {
  // 비동기 처리를 수행하는 코드
  const result = await Promise.resolve('비동기 처리 결과');
  return result;
}

asyncFunction().then((result) => {
  // 비동기 처리 결과를 처리합니다.
  console.log(result);
}).catch((error) => {
  // 에러를 처리합니다.
  console.error(error);
});
```

async-await을 사용하는 방법

1. 함수 앞에 `async` 키워드를 붙입니다.
2. 함수 내부에서 비동기 처리를 수행하는 코드 앞에 `await` 키워드를 붙입니다.
3. 비동기 처리 결과를 반환합니다.

async/await 문법을 이용하여 비동기 작업을 처리하면 코드의 가독성이 좋아지고 유지보수가 용이해집니다. 하지만 async/await 문법을 이용하여 비동기 작업을 처리하는 경우, 함수 내에서 `await` 키워드를 이용하여 기다리는 시간이 길어질 경우 브라우저가 먹통이 될 수 있습니다. 따라서 이를 해결하기 위해 비동기 작업을 작은 단위로 나누어 처리하는 것이 좋습니다.