Callback
Promise
async, await

mashup 노드팀 10분 세미나

Callback

Callback

자바스크립트에서 **비동기 처리**를 위해 사용하는 **함수**

비동기 처리

특정 코드의 연산이 끝날 때까지 기다려주지 않고 다음 코드를 실행하는 처리 방식

순차적으로 실행되지 않는 게 포인트

Callback을 쓰면

데이터가 준비된 시점에서 원하는 동작을 출력할 수 있다.

비동기만

아아메 준ㅂ...

안녕히 계세요!

콜백함수 사용

아아메 준비되었습니다!

어 가지고 가야지

사용 전

Title

```
function getData() {
    var tableData;
    $.get('https://domain.com/products/1', function(response) {
        tableData = response;
    });
    return tableData;
}

console.log(getData()); // undefined
```

사용 후

Title

```
function getData(callbackFunc) {
    $.get('https://domain.com/products/1', function(response) {
        callbackFunc(response);
        // 서버에서 받은 데이터 response를 callbackFunc() 함수에 넘겨줌
    });
}

getData(function(tableData) {
    console.log(tableData);
    // $.get()의 response 값이 tableData에 전달됨
});
```

근데 왜 문제?

콜백 지옥

콜백 안에 콜백을 무는 형식.
가독성이 떨어질 뿐더러 로직을 변경하기도 어렵다.

Title

```
const getCoffeeList = () => {
    setTimeout(() => {
            console.log('Espresso');
            setTimeout(() => {
                    console.log('Americano');
                    setTimeout(() => {
                            console.log('Latte');
                        },
                        500,
                    );
                },
                500,
            );
        },
        500,
    );
}

getCoffeeList();

// Espresso
// Americano
// Latte
```

기명 함수 Promise async await

기명 함수

기명 함수

익명 함수를 기명 함수로 변경하여 분리

장점

코드의 가독성을 높일 수 있고,
순서대로 읽는 게 가능

단점

일회성 함수를 전부 변수에 할당해야 한다는 번거로움

Title

```
function parseValueDone(id) {
    auth(id, authDone);
}
function authDone(result) {
    display(result, displayDone);
}
function displayDone(text) {
    console.log(text);
}
$.get('url', function(response) {
    parseValue(response, parseValueDone);
});
```

비교

Title

```
const getCoffeeList = () => {
    setTimeout(() => {
            console.log('Espresso');
            setTimeout(() => {
                    console.log('Americano');
                    setTimeout(() => {
                            console.log('Latte');
                        },
                        500,
                    );
                },
                500,
            );
        },
        500,
    );
}

getCoffeeList();

// Espresso
// Americano
// Latte
```

Title

```
const addEspresso = () => {
  console.log('Espresso');
  setTimeout(addAmericano, 500);
};

const addAmericano = () => {
  console.log('Americano');
  setTimeout(addLatte, 500);
};

const addLatte = () => {
  console.log('Latte');
};

const getCoffeeList = () => {
    setTimeout(addEspresso, 500);
}

getCoffeeList();

// Espresso
// Americano
// Latte
```

Promise

Promise

ES6에서 추가

대기, 이행, 거부의 3가지 상태

콜백 함수 내부에 resolve 또는 reject 함수를 호출하는 구문이 있을 경우,
둘 중 하나가 실행되기 전까지는
then 또는 catch 로 넘어가지 않음

비동기 작업의 동기적 표현이 가능

Title

```
function getData() {
  return new Promise(function(resolve, reject) {
    $.get('url 주소/products/1', function(response) {
      if (response) {
        resolve(response);
      }
      reject(new Error("Request is failed"));
    });
  });
}

// 위 $.get() 호출 결과에 따라 'response' 또는 'Error' 출력
getData().then(function(data) {
  console.log(data); // response 값 출력
}).catch(function(err) {
  console.error(err); // Error 출력
});
```

비교

Title

```
const addEspresso = () => {
  console.log('Espresso');
  setTimeout(addAmericano, 500);
};

const addAmericano = () => {
  console.log('Americano');
  setTimeout(addLatte, 500);
};

const addLatte = () => {
  console.log('Latte');
};

const getCoffeeList = () => {
    setTimeout(addEspresso, 500);
}

getCoffeeList();

// Espresso
// Americano
// Latte
```

Title

```
const getEspresso = () => {
  return new Promise((resolve) => {
    setTimeout(() => {
        console.log('Espresso');
        resolve('Espresso');
      },
      500,
    );
  });
}
const getAmericano = () => {
  return new Promise((resolve) => {
      // .5초 뒤에 'Americano' 찍는 로직
  });
}
const getLatte = () => {
  return new Promise((resolve) => {
      // .5초 뒤에 'Latte' 찍는 로직
  });
}

const getCoffeeList = () => {
  getEspresso()
    .then(getAmericano)
    .then(getLatte);
}

getCoffeeList();

// Espresso
// Americano
// Latte
```

Promise 정리

장점

비동기를 동기처럼 사용할 수 있음
개수가 많아져도 코드의 깊이가 깊어지지 않음

단점

에러를 잡을 때 몇 번째에서 발생했는지 알아내기 어려움
특정 값을 공유해가며 작업을 처리하는 번거로움

async / await

async / await

ES8에서 추가

함수의 앞에 async를 붙이고
함수의 내부 로직 중 비동기 처리 코드
앞에 async를 붙이는 형태

try catch로 예외 처리

Title

```
async function logTodoTitle() {
  try {
    var user = await fetchUser();
    if (user.id === 1) {
      var todo = await fetchTodo();
      console.log(todo.title); // delectus aut autem
    }
  } catch (error) {
    console.log(error);
  }
}
```

Title

```
const getEspresso = () => {
  return new Promise((resolve) => {
    setTimeout(() => {
        console.log('Espresso');
        resolve('Espresso');
      },
      500,
    );
  });
}
const getAmericano = () => {
  return new Promise((resolve) => {
      // .5초 뒤에 'Americano' 찍는 로직
  });
}
const getLatte = () => {
  return new Promise((resolve) => {
      // .5초 뒤에 'Latte' 찍는 로직
  });
}

const getCoffeeList = () => {
  getEspresso()
    .then(getAmericano)
    .then(getLatte);
}

getCoffeeList();

// Espresso
// Americano
// Latte
```

비교

Title

```
const getEspresso = () => {
  return new Promise((resolve) => {
    setTimeout(() => {
        console.log('Espresso');
        resolve('Espresso');
      },
      500,
    );
  });
}
const getAmericano = () => {
  return new Promise((resolve) => {
      // .5초 뒤에 'Americano' 찍는 로직
  });
}
const getLatte = () => {
  return new Promise((resolve) => {
      // .5초 뒤에 'Latte' 찍는 로직
  });
}

const getCoffeeList = async () => {
    await getEspresso();
    await getAmericano();
    await getLatte();
}

getCoffeeList();

// Espresso
// Americano
// Latte
```

async / await 정리

장점 콜백 함수와 프로미스의 단점을 보완하고 가독성을 높임

단점? 비동기로 실행될 Promise가 있다면 async함수 안에 항상 await을 써야한다

비교

```
const getCoffeeList = () => {
    setTimeout(() => {
            console.log('Espresso');
            setTimeout(() => {
                    console.log('Americano');
                    setTimeout(() => {
                            console.log('Latte');
                        },
                        500,
                    );
                },
                500,
            );
        },
        500,
    );
}

getCoffeeList();

// Espresso
// Americano
// Latte
```

```
const addEspresso = () => {
  console.log('Espresso');
  setTimeout(addAmericano, 500);
};

const addAmericano = () => {
  console.log('Americano');
  setTimeout(addLatte, 500);
};

const addLatte = () => {
  console.log('Latte');
};

const getCoffeeList = () => {
    setTimeout(addEspresso, 500);
}

getCoffeeList();

// Espresso
// Americano
// Latte
```

```
const getEspresso = () => {
  return new Promise((resolve) => {
    setTimeout(() => {
        console.log('Espresso');
        resolve('Espresso');
      },
      500,
    );
  });
}
const getAmericano = () => {
  return new Promise((resolve) => {
      // .5초 뒤에 'Americano' 찍는 로직
  });
}
const getLatte = () => {
  return new Promise((resolve) => {
      // .5초 뒤에 'Latte' 찍는 로직
  });
}

const getCoffeeList = () => {
  getEspresso()
    .then(getAmericano)
    .then(getLatte);
}

getCoffeeList();

// Espresso
// Americano
// Latte
```

```
const getCoffeeList = async () => {
    await getEspresso();
    await getAmericano();
    await getLatte();
}
```

감사합니다

Refer.

https://learnjs.vlpt.us/async/01-promise.html

https://joshua1988.github.io/web-development/javascript/javascript-asynchronous-operation/

https://velog.io/@cyranocoding/2019-08-02-1808-

%EC%9E%91%EC%84%B1%EB%90%A8-5hjytwqpqj