자바스크립트에서 비동기 처리

□ 학습내용

자바스크립트 비동기 소개

콜백 함수 소개

Promise 객체

async/await 구문

1 자바스크립트 비동기 소개

- 비동기(Asynchronous) 프로그래밍에서는 동시에 작업을 실행할 수 있다
- 자바스크립트는 싱글 스레드(한번에 하나의 명령만 수행)이지만, 콜백, 프로미스, 어싱크 어웨이트로 비동기 처리가 가능
- 콜백은 함수의 파라미터로 함수를 전달하고 처리가 끝나면 넘겨준 함수 실행. 가독성이 좋지 않음
- 프로미스(Promise)는 비동기 작업이 완료되면 결과를 반환하는 객체
  - 프로미스 객체는 상태를 가지고 있으며 작업완료시 성공 혹은 실패 상태가 됨
- 어싱크 어웨이트(async, await)는 프로미스를 사용하는 비동기 작업을 동기형태의 코드로 작성할 수 있게 해줌

2 콜백 함수 소개

- 비동기 코드를 원하는 순서로 실행하는 방법 중 가장 일반적인 방법
- 함수를 인자로 넘기고 작업완료시 실행
- 커피숍에서 점원에게 주문 후, 완료되면 점원이 손님을 호출하는 형태와 유사

2 콜백 함수 소개 (콜백 함수 작성해보기)

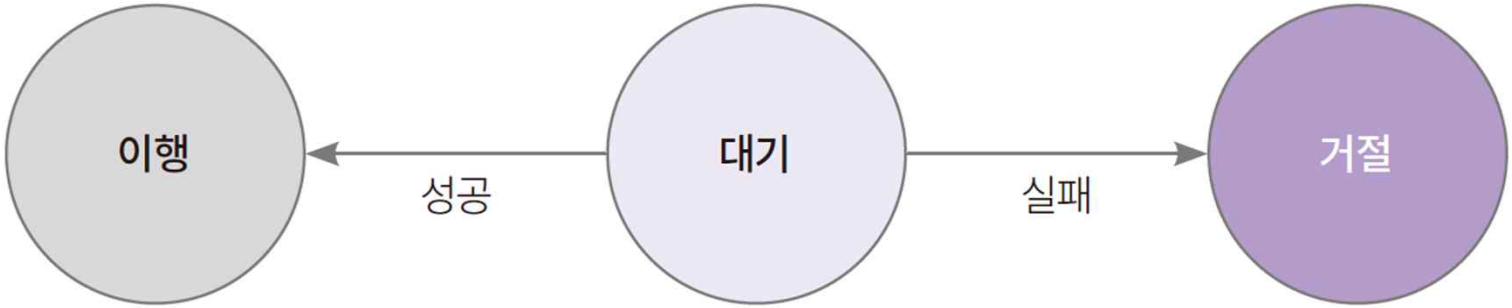
- 콜백이 3중으로 중첩된 함수

```
function register(user) {
  return saveDB(user, function (user) {
    return sendEmail(user, function (user) {
      return getResult(user);
    });
  });
}
```

3 Promise 객체

- Promise는 ' 이 코드는 미래의 어느 시점에 실행할 것이다' 라는 약속하는 객체
- Promise는 이행, 거절, 대기 세 가지 상태를 가질 수 있음
- 객체이므로 new 연산자로 인스턴스를 생성 할 수 있다

▼ Promise의 상태 변경



3 Promise 객체 (프로미스 코드 작성)

```
1  function registerByPromise(user) {
2    const result = saveDB(user)
3                     .then(sendEmail)
4                     .then(getResult);
5    console.log(result);
6    return result;
7  }
8
9  const myUser = { email: "andy@test.com", password: "1234", name: "andy" };
10
11 const result = registerByPromise(myUser);
12 result.then(console.log);
```

3 Promise 객체

3.1 동시에 여러 Promise 객체 호출하기

- Promise.all() 을 사용

```
Promise.all([Promise1, Promise2, … PromiseN])
```

- 기존코드를 변경 해보기

```
const myUser = { email: "andy@test.com", password: "1234", name: "andy" };
allResult = Promise.all([saveDB(myUser), sendEmail(myUser), getResult(myUser)]);
allResult.then(console.log);
```

3 Promise 객체

3.2 Promise 예외 처리하기

```
1  function registerByPromise(user) {
2    const result = saveDB(user)
3                     .then(sendEmail)
4                     .then(getResult)
5                     .catch(error => new Error(error))
6                     .finally(() => console.log("완료!"));
7    console.log(result);
8    return result;
9  }
10
11 const myUser = { email: "andy@test.com", password: "1234", name: "andy" };
12
13 const result = registerByPromise(myUser);
14 result.then(console.log);
```

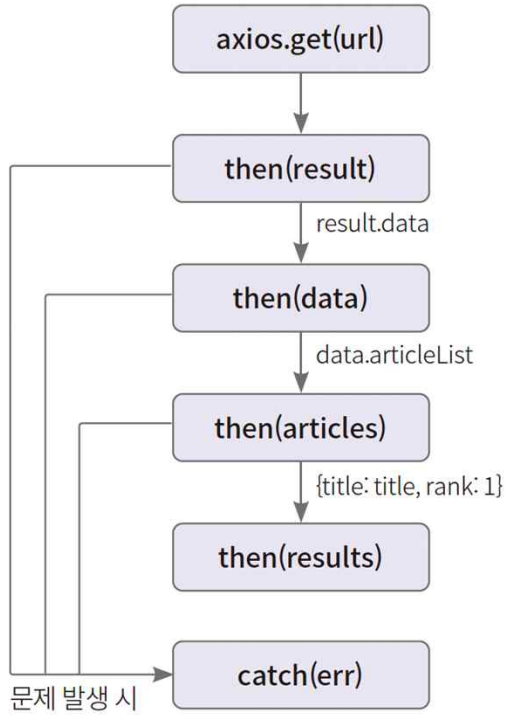
예외처리

성공/실패와 무관하게 실행

3 Promise 객체

3.3 복잡한 프로미스 예제 (현재 영화 상영 순위 TOP 20)

```
1  const axios = require("axios");
2  const url = "orl.kr/h9F";
3
4  axios
5    .get(url)
6    .then((result) => {
7      if (result.data) {
8        return result.data;    }})
9    .then((data) => data.articleList)
10   .then((articles) => {
11     return articles.map((article, idx) => {
12       return { title: article.title, rank: idx + 1 };
13     });
14   })
15   .then((results) => {
16     for (let movieInfo of results) {
17       console.log(`[${movieInfo.rank}위] ${movieInfo.title}`);
18     }})
19   .catch((err) => {
20     console.log("<<에러발생>>", err);
21   });
```



3 Promise객체

프로미스의 문제점 및 대안

- Promise에 콜백 함수를 넘기면 안된다.

```
1  function myWork(work) {
2      return new Promise((resolve, reject) => {
3          if (work === 'done') {
4              resolve('게임 가능');
5          } else {
6              reject(new Error("게임 불가능"));
7          }
8      })
9  }
10
11 myWork('done').then(function (value) { console.log(value) }, function (err) { console.error(err) });
12
13 myWork('doing')
14     .then(function (value) { console.log(value) })
15     .catch(function (err) { console.error(err) });
```

Promise에 콜백
함수를 넘기면 안됨

then() 으로 처리

3 Promise객체

프로미스의 문제점 및 대안

- 프로미스를 중첩해서 사용해서는 안됨

```
1  myWork('done')
2      .then(function (result) {
3          playGame(result).then(function (val) {
4              console.log(val);
5          });
6      })
```

- then() 으로 연결

```
1  myWork('done')
2  .then(playGame)
3  .then(console.log)
```

4 async await 구문

- 가장 최근에 도입된 비동기 처리 방식
- 콜백과 프로미스의 단점을 보완
- 가독성 높은 코드 작성 가능
- async는 함수 앞에 붙인다
- async가 붙은 함수는 Promise를 반환한다
- 내부적으로는 제너레이터를 사용한다

4 async await 구문

- async await 예제

```
1  async function showName() {
2    const name = await myName();
3    console.log(name);
4  }
5  console.log(showName());
```

4 async await 구문

- async await 를 사용하여 1부터 10까지 1초마다 출력하는 예제

```
1  function waitOneSecond(msg) {
2    return new Promise((resolve, _) => {
3      setTimeout(() => resolve(`${msg}`), 1000);
4    });
5  }
6
7  async function countOneToTen() {
8    for (let x of [...Array(10).keys()]) {
9      let result = await waitOneSecond(`${x + 1}초 대기중...`);
10     console.log(result);
11   }
12   console.log("완료");
13 }
14
15 countOneToTen();
```

callback, promise, async await 비교

▼ callback, promise, async await 비교

구분	callback	promise	async/await
에러 처리	콜백 함수 내에서 처리	catch() 메서드로 처리	try-catch 블록으로 처리
가독성	간단한 경우에는 괜찮으나, 점점 복잡해짐	가독성 좋음	가독성 좋음
중첩 처리	콜백 함수 내에서 처리	then() 메서드를 사용	await 키워드를 사용
예시 코드	`function getData(callback) {   const data = { age: 30, name: "andy" };   callback(data); }  getData((data) => {   console.log(data); });`	`const dataPromise = Promise.resolve({ age: 30, name: "andy" });  dataPromise   .then((data) => {   console.log(data);   })   .catch((error) => {   console.log(error);   });`	`const dataPromise = Promise.resolve({ age: 30, name: "andy" });  async function main() {   try {   const data = await dataPromise;   console.log(data);   } catch (error) {   console.log(error);   } }  main();`