JS-중급 15

태그	
날짜	@2023년 6월 7일

```
async & await :
promise의 then 메소드를 체인형식으로 호출하는 것보다
가독성이 좋아진다.

async function getName(){
  return "Mike";
}

console.log(getName());

async 키워드를 앞에 붙히면 항상 promise를 반환한다.
// [object Promise]
{}

그러므로 함수를 사용하고 .then을 사용할 수 있다.

async function getName(){
  return "Mike";
}

getName().then((name)=>{
  console.log(name);
});
"Mike"

만약 반환값이 프로미스라면

async function getName(){
  return Promise.resolve("Tom");
}

getName().then((name)=>{
  console.log(name);
});
"Tom"

"Tom"을 그대로 사용한다.
그리고 함수 내부에서 예외가 발생하면 reject 상태의 프로미스가
반환된다.
async function getName(){
  // return Promise.resolve("Tom");
  throw new Error("err...");
```

```
}

// getName().then((name)=>{
//   console.log(name);
// });
getName().catch((err)=>{
  console.log(err);
})
// [object Error]
{}
```

```
await : async 함수 내부에서만 사용할 수 있다.

function getName(name) {
  return new Promise((resolve, reject)=> {
    setTimeout(()=>{
      resolve(name);
    },1000);
  });
}

async function showName(){
  const result = await getName('Mike');
  console.log(result);
}

console.log("시작");
showName(); // 1초 뒤 실행

"시작"
"Mike"
```

```
const f1 = () =>{
  return new Promise((res, rej)=>{
    setTimeout(()=>{
      res("1번 주문 완료");
    },1000)
  });
};
const f2 = (message) =>{
```

```
  console.log(message);
  return new Promise((res, rej)=>{
    setTimeout(()=>{
        res("2번 주문 완료");
    },3000)
  });
};
const f3 = (message) =>{
  console.log(message);
  return new Promise((res, rej)=>{
    setTimeout(()=>{
      res("3번 주문 완료");
    },2000)
  });
};

console.log("시작");
async function order(){
  const result1 = await f1();
  const result2 = await f2(result1);
  const result3 = await f3(result2);
  console.log(result3);
  console.log("종료");
}
order();

"시작"
"1번 주문 완료"
"2번 주문 완료"
"3번 주문 완료"
"종료"

중간에 에러가 발생하면 코드가 멈춘다.
promise의 경우 catch를 사용했지만
async await 의 경우 try catch 문으로 감싸주면 된다.

const f1 = () =>{
  return new Promise((res, rej)=>{
    setTimeout(()=>{
      res("1번 주문 완료");
    },1000)
  });
};
const f2 = (message) =>{
  console.log(message);
  return new Promise((res, rej)=>{
    setTimeout(()=>{
        rej("xxx");
    },3000)
  });
};
const f3 = (message) =>{
  console.log(message);
  return new Promise((res, rej)=>{
```

```
    setTimeout(()=>{
      res("3번 주문 완료");
    },2000)
  });
};

console.log("시작");
async function order(){
  try{
  const result1 = await f1();
  const result2 = await f2(result1);
  const result3 = await f3(result2);
  console.log(result3);
  }catch(e){
  console.log(e);
  }
  console.log("종료");
}
order();


"시작"
"1번 주문 완료"
"xxx"
"종료"
```

```
const f1 = () =>{
  return new Promise((res, rej)=>{
    setTimeout(()=>{
      res("1번 주문 완료");
    },1000)
  });
};
const f2 = (message) =>{
  console.log(message);
  return new Promise((res, rej)=>{
    setTimeout(()=>{
        res("2번 주문 완료");
    },3000)
  });
};
const f3 = (message) =>{
  console.log(message);
  return new Promise((res, rej)=>{
    setTimeout(()=>{
      res("3번 주문 완료");
    },2000)
```

```
  });
};

console.log("시작");
async function order(){
  try{
  const result = await Promise.all([f1(),f2(),f3()]);
  console.log(result);
  }catch(e){
  console.log(e);
  }
  console.log("종료");
}
order();

// [object Array] (3)
["1번 주문 완료","2번 주문 완료","3번 주문 완료"]
"종료"
```