**promise**

* 비동기 작업을 조금 더 편하게 처리해주는 ES6 기능
* 이전엔 비동기처리를 할 때 전부 콜백함수로 처리해야했다.
* 이는 비동기작업이 중접될경우 코드가 쉽게 난잡해짐
* 이를 콜백지옥이라 부름.
* promise를 사용하면, 비동기적으로 작업을 수행하고, 작업이 완료됐을 때 성공/실패를 반환하므로 콜백지옥에 빠지지 않고 코드를 깔끔하게 작성할 수 있다.

**promise 만들기**

* 다음처럼 new로 만듬

const myPromise = new Promise((resolve, reject) => {

…

}

* 성공할 때는 resolve를 호출하고, 실패할 때는 reject를 호출해주면 됨

//성공시

const myPromise = new Promise((resolve, reject) => {

setTimeout(() => {

resolve(1);

}, 1000);

});

//실패구현

const myPromise = new Promise((resolve, reject) => {

setTimeout(() => {

reject(new Error());

}, 1000);

});

//성공과 실패는 각각 then과 catch로 받을 수 있음

myPromise

.then(n => {

console.log(n);

})

.catch(error => {

console.log(error);

});

* Promise 함수 구현

function increaseAndPrint(n) {

return new Promise((resolve, reject) => {

setTimeout(() => {

const value = n+1;

if (value === 5) {

const error = new Error();

error.name = “ValueIsFiveError”;

reject(error);

return;

}

console.log(value);

resolve(value);

}, 1000);

});

}

increaseAndPrint(0).then((n) => {

console.log(“result : “, n);

});

* 위 함수 응용

성공시 resolve가 반환되는 점에서 콜백지옥을 벗어날 수 있다

increaseAndPrint(0)

.then(n => {

return increaseAndPrint(n);

})

.then(n => {

return increaseAndPrint(n);

})

.then(n => {

return increaseAndPrint(n);

})

.then(n => {

return increaseAndPrint(n);

})

.then(n => {

return increaseAndPrint(n);

})

.catch(e=> {

console.error(e);

});

* 성공할 때마다 반환하고 다음이 실행되므로, 연속 콜백을 사용하기에 적합하다.
* 위코드는 아래 코드로 정리 가능하다

increaseAndPrint(0)

.then(increaseAndPrint)

.then(increaseAndPrint)

.then(increaseAndPrint)

.then(increaseAndPrint)

.then(increaseAndPrint)

.catch(e => {

console.error(e);

});