

# JS-중급 14

≡ 태그	
📅 날짜	@2023년 6월 7일

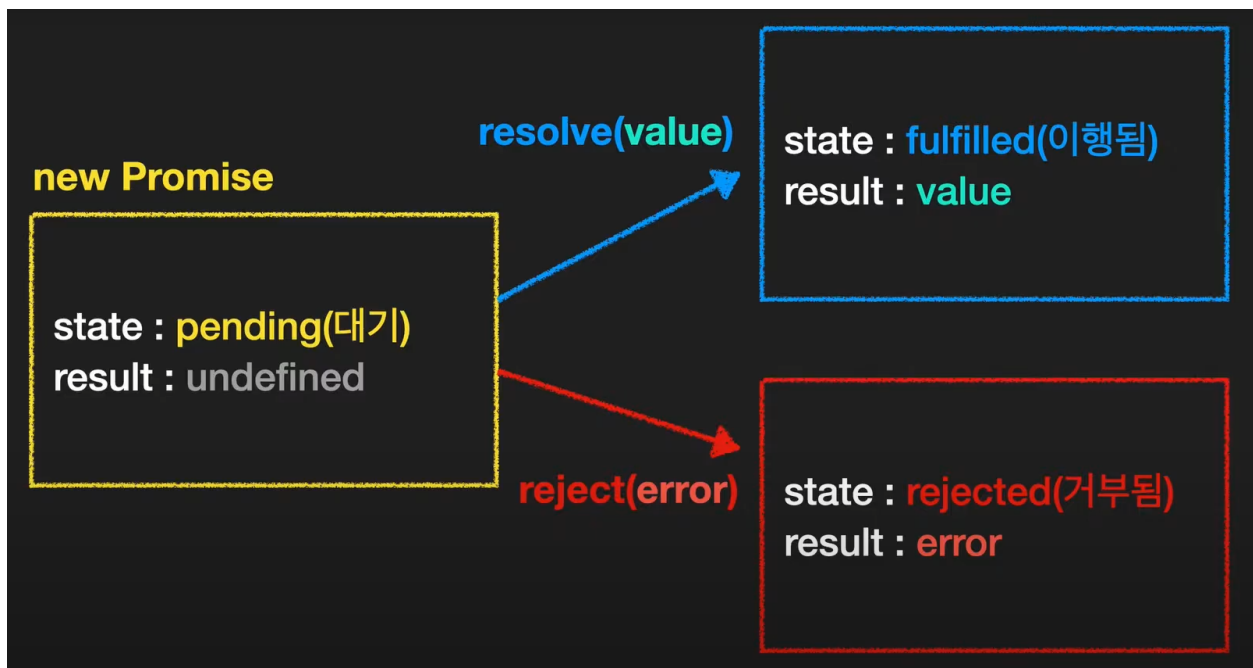
Promise :

```
const pr = new Promise((resolve, reject) => {  
  // code  
});
```

resolve : 성공한 경우 실행되는 함수

reject : 실패한 경우 실행되는 함수

어떤일이 완료된 이후 실행되는 함수를 콜백함수라고 한다.



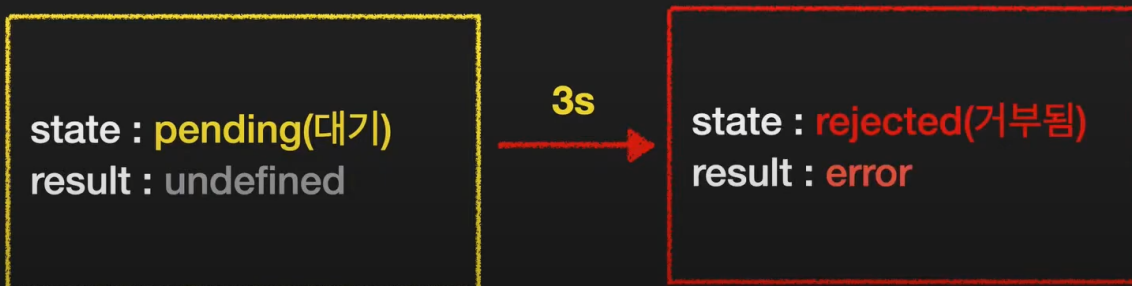
성공한 코드

```
const pr = new Promise((resolve, reject) => {
  setTimeout(()=>{
    resolve('OK')
  }, 3000)
});
```




실패

```
const pr = new Promise((resolve, reject) => {
  setTimeout(()=>{
    reject(new Error('error..'))
  }, 3000)
});
```



```
const pr = new Promise((resolve, reject) => {
  setTimeout(()=>{
    resolve('OK')
  }, 3000)
});

pr.then(
  function(result){}, — 이행 되었을때 실행
  function(err){} — 거부 되었을때 실행
);
```



```
const pr = new Promise((resolve, reject) => {
  setTimeout(()=>{
    resolve('OK')
  }, 3000)
});

pr.then(
  function(result){
    console.log(result + ' 가지러 가자. ');
  },
  function(err){
    console.log('다시 주문해주세요.. ');
  }
);
```

catch : reject인 경우에만 사용할 수 있음  
finally : 이행이든 거부든 코드가 완료되면 항상 실행된다.  
로딩화면같은 것을 없앨때 유용하다.

```
pr.then(  
  function(result){}  
).catch(  
  function(err){}  
).finally(  
  function(){  
    console.log('--- 주문 끝 ---')  
  }  
)
```

```
const pr = new Promise((resolve, reject) => {  
  setTimeout(()=>{  
    // resolve("ok");  
    reject(new Error("err..."))  
  }, 1000)  
});  
  
console.log("시작");  
pr.then((result) =>{  
  console.log(result);  
})  
  .catch((err) => {  
    console.log(err);  
  })  
  .finally(()=>{  
    console.log("끝");  
  });  
  
"시작"
```

```
// [object Error]
"끝"
```

```
const f1 = (callback) =>{
  setTimeout(function() {
    console.log("1번 주문 완료");
    callback();
  },1000);
};

const f2 = (callback) =>{
  setTimeout(function() {
    console.log("2번 주문 완료");
    callback();
  },1000);
};

const f3 = (callback) =>{
  setTimeout(function() {
    console.log("3번 주문 완료");
    callback();
  },1000);
};

console.log("시작")
f1(function(){
  f2(function(){
    f3(function(){
      console.log("끝")
    })
  })
})
// callback hell 의 예시
```

-----

promise사용

```
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)
  });
};

// 프로미스 체이징(Promise chaining)
console.log("시작");
f1()
  .then((res) => f2(res))
  .then((res) => f3(res))
  .then((res) => console.log(res))
  .catch(console.log)
  .finally(()=>{
    console.log("끝!");
  });

"시작"
"1번 주문 완료"
"2번 주문 완료"
"3번 주문 완료"
"끝!"

```

```

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)
  });
};

```

```

    },2000)
  });
};

console.log("시작");
f1()
  .then((res) => f2(res))
  .then((res) => f3(res))
  .then((res) => console.log(res))
  .catch(console.log)
  .finally(()=>{
    console.log("끝!");
  });
"시작"
"1번 주문 완료"
"xxx"
"끝!"

-----
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)
  });
};

//Promise.all
Promise.all([f1(),f2(),f3()]).then((res) => {
  console.log(res);
});

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

```

Promise.race :

all 은 모든 작업이 완료될때까지 기다리지만  
race는 하나라도 1등으로 완료되면 끝낸다.

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

//Promise.race
Promise.race([f1(),f2(),f3()]).then((res) => {
  console.log(res);
});

"1번 주문 완료"
```