promise

ECMAScript 2015

S67

일정 : 2016/9/22 - 16/10/27
시간 : 매주 목요일 저녁 8-10시
장소 : 네이버 D2 STARTUP FACTORY

projectB5 x Bsidesoft x RoasteryKay x NAVER D2

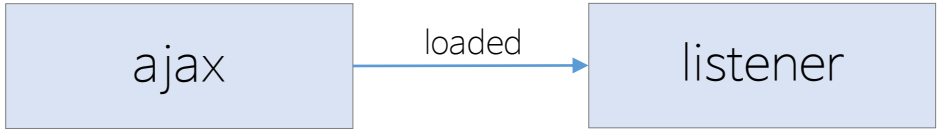
thenable의 의미

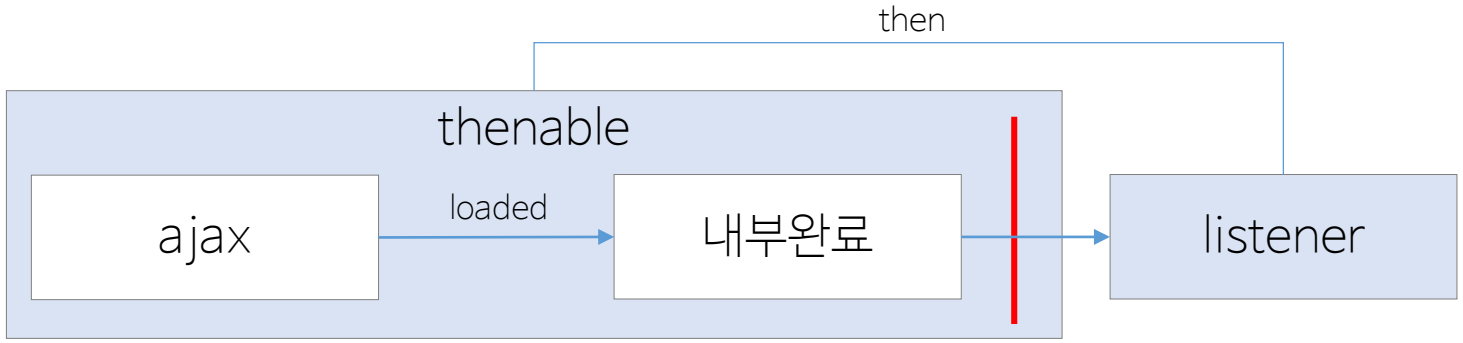
기존의 모든 비동기처리는 즉발성이므로 비동기를 통보하는 쪽이 주도권을 쥐고 있음.
수신하는 쪽에서 원할 때 처리하려면 비동기를 시작하는 행위 자체를 객체화할 필요가 있음.

이러한 포괄적인 비동기행위에 대한 객체정의에 적용되는 프로토콜이 thenable임

```
//기존
ajax.load(url, listener);

//thenable
let thenable = ajax.load(url);
..
..
thenable.then(listener);
```





then method

thenable.then(v);

반환값 : Promise
인자:
호출가능하지 않은 객체인 경우
function(v) {return v;} 로 대체
호출가능한 객체가 Promise를 반환하지 않는 경우
Promise.resolve(v); 로 변환되어 처리됨.

25.4.5.3.1 PerformPromiseThen (promise, onFulfilled, onReje

The abstract operation PerformPromiseThen performs the
onRejected as its settlement actions. The result is *resultCapab*

1. Assert: IsPromise(*promise*) is **true**.
2. Assert: *resultCapability* is a PromiseCapability record.
3. If IsCallable(*onFulfilled*) is **false**, then
   a. Let *onFulfilled* be **"Identity"**.
4. If IsCallable(*onRejected*) is **false**, then
   a. Let *onRejected* be **"Thrower"**.

```
Promise.resolve(3).then(5).then(console.log); //3

Promise.resolve(3).then(v=>v+1).then(console.log);
Promise.resolve(3).then(v=>Promise.resolve(v+1)).then(console.log);
```

then chain

then이 Promise를 반환하므로 연속적인 then체인을 만들 수 있음.

1. 기존의 동기성 작업을
2. 비동기적인 체이닝형태의 커맨드 리스트로 변환하여
3. 흡사한 구조로 기술하되 비동기로 처리되게 할 수 있음.

```
//then 체인화
const gene = function*(arr){
    for(const v of arr) yield p=>v + p;
};
let p = Promise.resolve(0);

for(let v of gene([1,2,3,4])) p = p.then(v);

p.then(v=>console.log(v)); //10
```

```
//then 체인화
const gene = function*(arr){
    for(const v of arr) yield p=>new Promise(
            rev=>setTimeout(_=>rev(v + p), 500
        ));
};
let p = Promise.resolve(0);

for(let v of gene([1,2,3,4])) p = p.then(v);

p.then(v=>console.log(v)); //2초뒤 10
```

연속된 비동기작업의 처리

병렬적처리와 직렬적처리

병렬처리 - 동시에 비동기작업 여러개가 실행됨
직렬적처리 - 비동기작업이 순차적으로 실행됨

중단점 지정 - 병렬처리 중 일정 조건을 달성하는 지점에서 대기

```
//전역카운터
const cnt = function*(start, end) {
    for(let i = start; i <= end; i++) yield i;
};

const g = function*(g, f, interval) {
    for(const i of g) yield _=>new Promise(rev=>(f(i), setTimeout(rev, interval)));
};

let p = Promise.resolve();
for(const v of g(cnt(0, 3), v=>console.log(v), 1000)) p = p.then(v);

const el = document.getElementById('target');
const f =v=>el.style.marginLeft = v + 'px';
p = Promise.resolve();
for(const v of g(cnt(0, 500), f, 10)) p = p.then(v);
```

중단점처리

Promise.all 사용

```
const img = function*(g, f) {
    const imgs = [];
    for(const i of g) {
        yield _=>new Promise(res=>{
            const img = document.createElement('img');
            img.src = i + '.jpg';
            imgs.push(img);
            img.onload=_=>setTimeout(_=>(f(img), res(imgs)), 200);
        });
    }
};

const unwrapper = function*(g) {for(const f of g) yield f(); };

Promise.all(unwrapper(img(cnt(1,5),console.log)))
    .then(v=>v[0].reduce(
        (p,el)=>(p.appendChild(el), p),
        document.getElementById('stage')
    ));
```