ECMAScript 2015

- ES2015를 사용하기 위한 프로젝트 설정
- let과 const
- 기본 파라미터와 가변 파라미터
- 구조분해 할당(destructuring assignment)
- 화살표 함수(Arrow function)
- 새로운 객체 리터럴
- 템플릿 리터럴
- 컬렉션
- 클래스
- 모듈
- Promise

개요

ECMAScript 2015

- A.K.A. ES2015, ES6
- 이전 버전의 자바스크립트로 번역(Transpile)
  - babel 또는 typescript, coffescript 등을 이용하여 번역
- SPA(Single Page Application) 개발을 위해 vuex, vue-router 등의 요소가 사용되어야 하며 이를 위해 ES6 또는 typescript 의 사용이 필요함.
- 이 책에서는 ES6 코드를 babel을 이용해 트랜스파일할 것임.

ES2015를 사용하기 위한 프로젝트 설정

- 디렉토리 생성 후 다음 명령 수행
  - npm init --> package.json 파일 생성
  - 전역 수준에서
    - npm install -g babel-cli babel-preset-es2015
  - 프로젝트 수준에서
    - npm install --save-dev babel-cli babel-preset-es2015
  - 결과 : node_modules 디렉토리와 package.json의 변화

```
{
  "name": "es2015",
  "version": "1.0.0",
……
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "babel-cli": "^6.24.1",
    "babel-preset-es2015": "^6.24.1"
  }
}
```

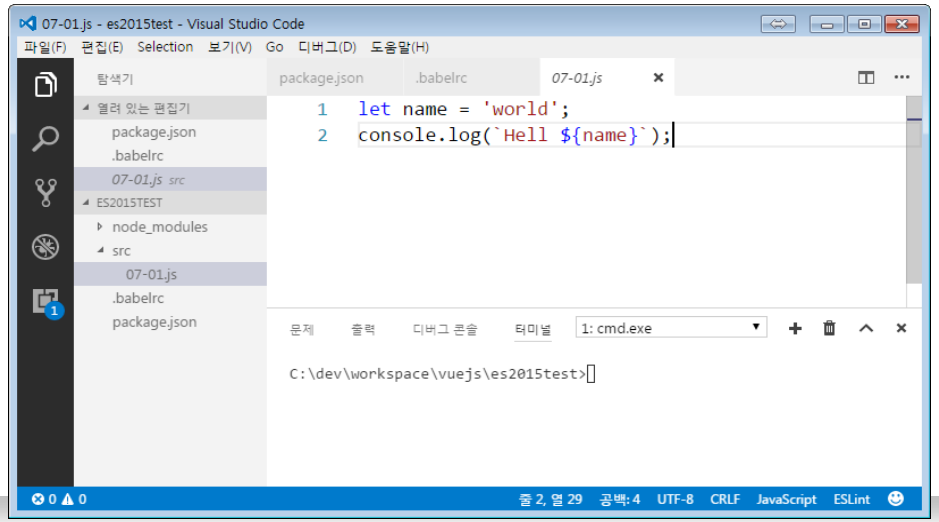
ES2015를 사용하기 위한 프로젝트 설정

.babelrc 파일 작성

```
{
   "presets" : [ "es2015"]
}
```

기능의 작동 여부 확인

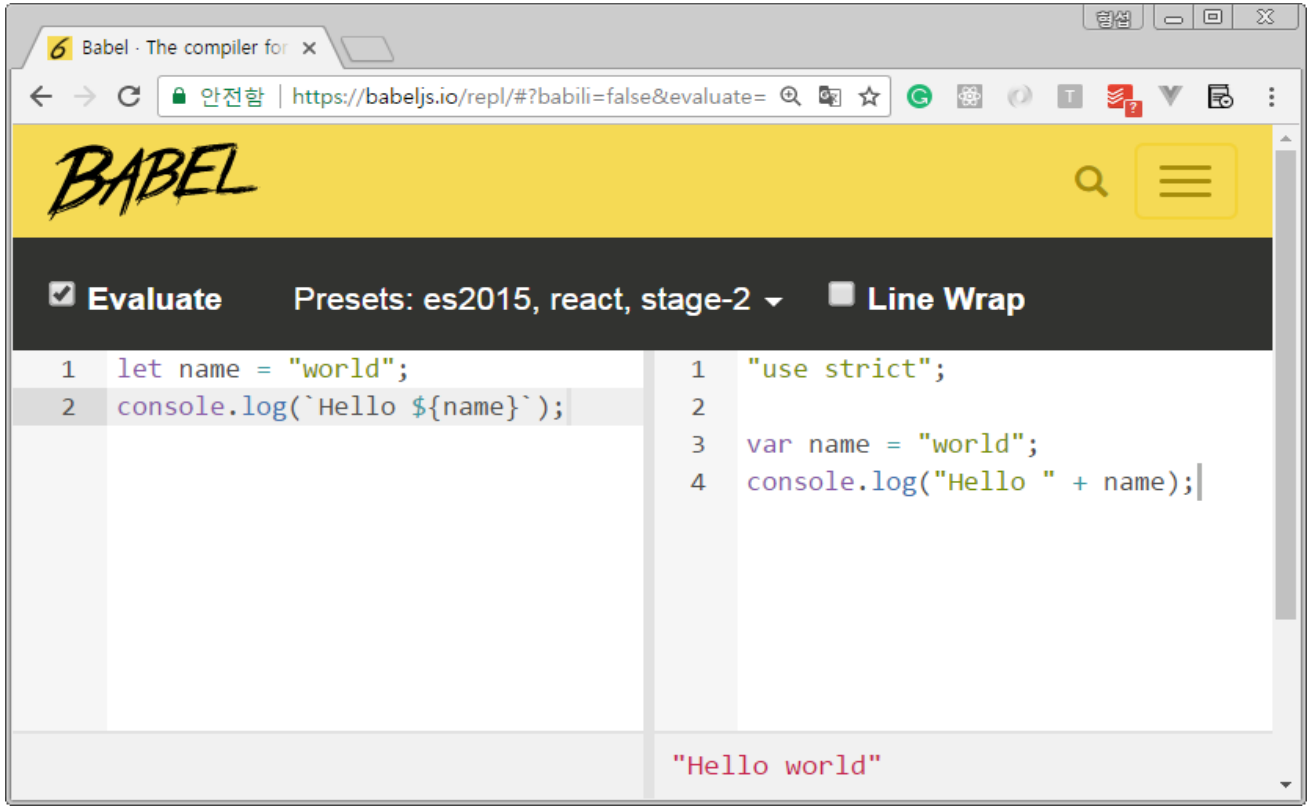
- src/07-01.js 작성후 다음 명령 실행
- babel src -d build
- 트랜스파일된 코드 실행
  - node bui



ES2015를 사용하기 위한 프로젝트 설정

babel REPL 도구

- 브라우저를 이용해 온라인 상에서 직접 트랜스파일할 수 있음



let과 const

var 키워드

- hoisting
  - 코드가 실행되기 전에 변수를 미리 생성함 --> 변수를 중복 선언해도 오류 발생(X)
- 블록 단위의 스코프 지원(X).
  - 함수 단위의 스코드만 지원

let 키워드

- Hoisting 하지 않음
- 블록 단위의 스코프 지원

```
let msg = "GLOBAL";
function outer(a) {
    let msg = "OUTER";
    console.log(msg);
    if (true) {
        let msg = "BLOCK";
        console.log(msg);
    }
}
```

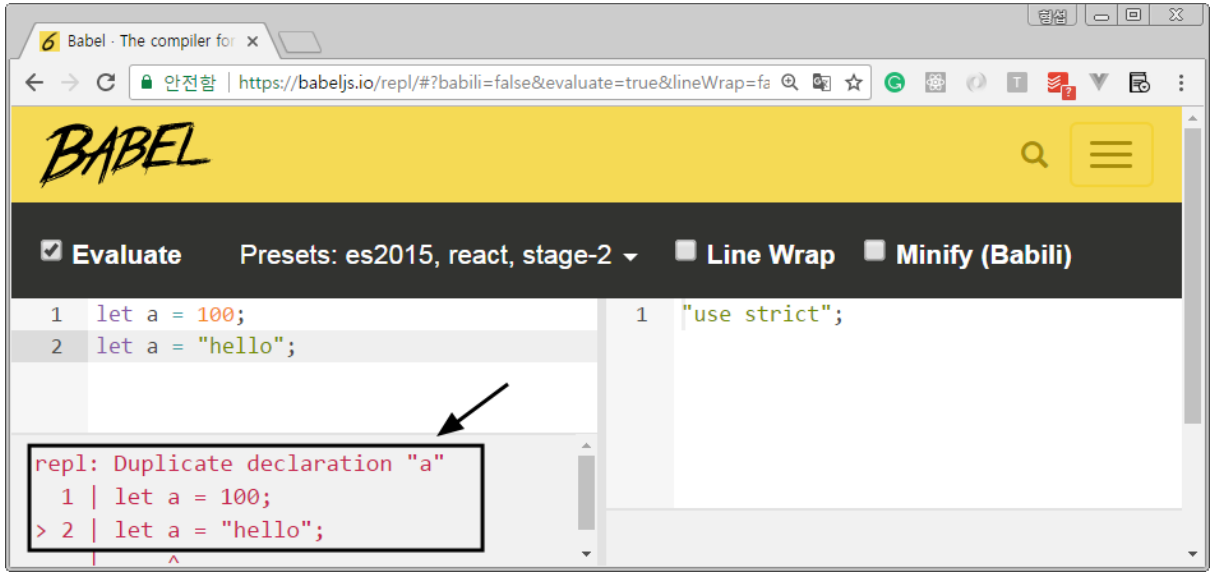
```
"use strict";

var msg = "GLOBAL";
function outer(a) {
    var msg = "OUTER";
    console.log(msg);
    if (true) {
        var _msg = "BLOCK";
        console.log(_msg);
    }
}
```

let과 const

let은 중복을 허용하지 않음

- var는 오류를 일으키지 않음

```
var a = 100;
var a = 'hello';
var a = { name:"홍길동", age:20 };
```



기본 파라미터와 가변 파라미터

기본 파라미터(Default Parameter)

- 함수 파라미터의 기본값을 지정할 수 있음

```
function addContact(name, mobile,
                    home="없음",
                    address="없음",
                    email="없음") {
   var str = `name=${name}, mobile=${mobile}, home=${home},
            address=${address}, email=${email}`;
   console.log(str);
}

addContact("홍길동", "010-222-3331")
addContact("이몽룡", "010-222-3331", "02-3422-9900", "서울시");
```

```
문제    출력    디버그 콘솔    터미널                    1: cmd.exe

C:\dev\workspace\vuejs\es2015test>node build/07-03.js
name=홍길동, mobile=010-222-3331, home=없음,
               address=없음, email=없음
name=이몽룡, mobile=010-222-3331, home=02-3422-9900,
               address=서울시, email=없음
```

기본 파라미터와 가변 파라미터

가변 파라미터(Rest Operator)

- 여러개 파라미터 값을 배열로 받을 수 있도록 함.

```
function foodReport(name, age, ...favoriteFoods) {
    console.log(name + ", " + age);
    console.log(favoriteFoods);
}

foodReport("이몽룡", 20, "짜장면", "냉면", "불고기");
foodReport("홍길동", 16, "초밥");
```

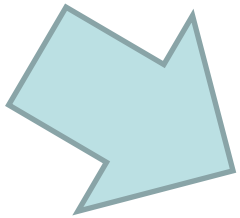
```
C:\dev\workspace\vuejs\es2015test>node build/07-04.js
이몽룡, 20
[ '짜장면', '냉면', '불고기' ]
홍길동, 16
[ '초밥' ]
```

구조 분해 할당

destructuring assignment

- 배열, 객체의 값들을 추출하여 여러 변수에 할당할 수 있는 기능

```
let arr = [10,20,30,40];
let [a1,a2,a3] = arr;
console.log(a1, a2, a3);

let p1 = {name:"홍길동", age:20, gender:"M"};
let { name:n, age:a, gender } = p1;
console.log(n,a,gender);
```



```
"use strict";

var arr = [10, 20, 30, 40];
var a1 = arr[0],
    a2 = arr[1],
    a3 = arr[2];

console.log(a1, a2, a3);

var p1 = { name: "홍길동", age: 20, gender: "M" };
var n = p1.name,
    a = p1.age,
    gender = p1.gender;

console.log(n, a, gender);
```

구조 분해 할당

함수의 파라미터에서도 사용 가능

```
function addContact({name, phone, email="이메일 없음", age=0}) {
    console.log("이름 : " + name);
    console.log("전번 : " + phone);
    console.log("이메일 : " + email);
    console.log("나이 : " + age);
}

addContact({
    name : "이몽룡",
    phone : "010-3434-8989"
})
```

```
C:\dev\workspace\vuejs\es2015test>node build/07-06.js
이름 : 이몽룡
전번 : 010-3434-8989
이메일 : 이메일 없음
나이 : 0
```

화살표 함수

Arrow Function

- 기존 함수 표현식에 비해 간결
- 함수를 정의하는 영역의 this를 그대로 전달받을 수 있음.
  - 기존 함수 표현식은 바깥쪽 스코프의 this를 전달받기 위해서 apply, call, bind 함수의 도움을 받아야 함.

```
var test1 = function(a,b) {
    return a+b;
}
let test2 = (a,b) =>{
    return a+b;
};
let test3 = (a,b) => a+b;

console.log(test1(3,4));
console.log(test2(3,4));
console.log(test3(3,4));
```

Evaluate | Presets: es2015, react, stage-2 | Line Wrap | Minify (Babili)

```
1  var test1 = function(a,b) {
2      return a+b;
3  }
4
5  let test2 = (a,b) =>{
6      return a+b;
7  };
8
9  let test3 = (a,b) => a+b;
10
11 console.log(test1(3,4));
12 console.log(test2(3,4));
13 console.log(test3(3,4));
```

```
1  "use strict";
2
3  var test1 = function test1(a, b) {
4      return a + b;
5  };
6
7  var test2 = function test2(a, b) {
8      return a + b;
9  };
10
11 var test3 = function test3(a, b) {
12     return a + b;
13 };
14
15 console.log(test1(3, 4));
16 console.log(test2(3, 4));
17 console.log(test3(3, 4));
```

화살표 함수

함수의 this

- 이전 함수에서의 this
  - 호출하는 문맥에 의해 좌우됨.
  - 문맥을 넘어서서 this를 연결하려면 bind, apply, call 등의 함수 수준의 메서드를 이용함

```
function Person(name, yearCount) {
    this.name = name;
    this.age = 0;

    var incrAge = function() {
        this.age++;
    }
    for (var i=1; i <= yearCount; i++) {
        incrAge();
    }
}
var p1 = new Person("홍길동",20);
//--this.age는 0이 출력됨.
console.log(p1.name + "님의 나이 : " + p1.age);
```

```
…
for (var i=1; i <= yearCount; i++) {
  incrAge.apply(this);
}
…
```

화살표 함수

함수의 this(이어서)

- 화살표 함수에서의 this
  - 화살표 함수를 둘러싸고 있는 영역의 this를 화살표 함수 내부로 전달함.

```
function Person(name, yearCount) {
    this.name = name;
    this.age = 0;
    var incrAge = ()=> {
        this.age++;
    }
    for (var i=1; i <= yearCount; i++) {
        incrAge();
    }
}
var p1 = new Person("홍길동",20);
//--this.age는 20이 출력됨.
console.log(p1.name + "님의 나이 : " + p1.age);
```

새로운 객체 리터럴

- ES2015에서 객체의 속성 표기법이 개선되었음
  - 객체의 속성명이 변수명과 동일하다면 생략 가능
  - 새로운 메서드 표기법

```
var name = "홍길동";
var age = 20;
var email = "gdhong@test.com";

//var obj = { name: name, age: age, email: email };
var obj = { name, age, email };
console.log(obj);
```

```
let p1 = {
    name : "아이패드",
    price : 200000,
    quantity : 2,
    order : function() {
        if (!this.amount) {
            this.amount = this.quantity * this.price;
        }
        console.log("주문금액 : " + this.amount);
    },
    discount(rate) {
        if (rate > 0 && rate < 0.8) {
            this.amount = (1-rate) * this.price * this.quantity;
        }
        console.log((100*rate) + "% 할인된 금액으로 구매합니다.");
    }
}
p1.discount(0.2);
p1.order();
```

템플릿 리터럴

Template Literal

- 역따옴표로 묶여진 문자열에서 템플릿 대입문을 이용해 문자열을 끼워넣을 수 있음
- 개행 문자를 포함하여 여러줄로 작성할 수 있음

```
var d1 = new Date();
var name = "홍길동";
var r1 = `${name} 님에게 ${d1.toDateString() }에 연락했다.`;
console.log(r1);

var product = "갤럭시S7";
var price = 199000;
var str = `${product}의 가격은
      ${price}원 입니다.`;
console.log(str);
```

컬렉션

- ES2015에서 Set, Map, WeakSet, WeakMap 제공
- Set
  - Union이나 Intersect와 같은 다양한 집합 연산 제공

```
var s1 = new Set();
s1.add("사과");   s1.add("배");
s1.add("사과");   s1.add("포도");
//실행 결과 : Set { '사과', '배', '포도' }
console.log(s1);

var john = new Set(["사과", "포도", "배"]);
var susan = new Set(["파인애플", "키위", "배"]);

//합집합 : Set { '사과', '포도', '배', '파인애플', '키위' }
var union = new Set([...john.values(), ...susan.values()]);
console.log(union);

//교집합 : Set { '배' }
var intersection = new Set([...john.values()].filter(e => susan.has(e)));
console.log(intersection);

//차집합 : Set { '사과', '포도' }
var diff = new Set([...john.values()].filter(e => !susan.has(e)));
console.log(diff);
```

```
문제   출력   디버그 콘솔   터미널                    1: cmd.exe

C:\dev\workspace\vuejs\es2015test>node build/07-12.js
Set { '사과', '배', '포도' }
Set { '사과', '포도', '배', '파인애플', '키위' }
Set { '배' }
Set { '사과', '포도' }
```

컬렉션

Map

- 키-값 쌍의 집합체. 키의 중복을 허용하지 않음

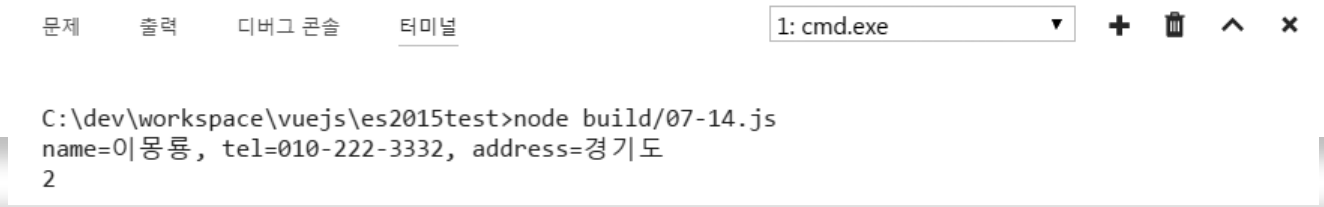
```
let teams = new Map();
teams.set('LG', '트윈스');      teams.set('삼성', '라이온스');
teams.set('NC', '다이노스');    teams.set('기아', '타이거스');
teams.set('한화', '이글즈');    teams.set('롯데', '자이언츠');

console.log(teams.has("SK"));        //false
console.log(teams.get("LG"));        //트윈스
```

```
문제    출력    디버그 콘솔    터미널                    1: cmd.exe

C:\dev\workspace\vuejs\es2015test>node build/07-13.js
false
트윈스
```

클래스

- 이전 버전까지는 클래스를 지원하지 않았음
  - 함수를 이용해 유사 클래스를 만드는 방법을 사용했음
- ES2015부터 공식적으로 클래스를 지원

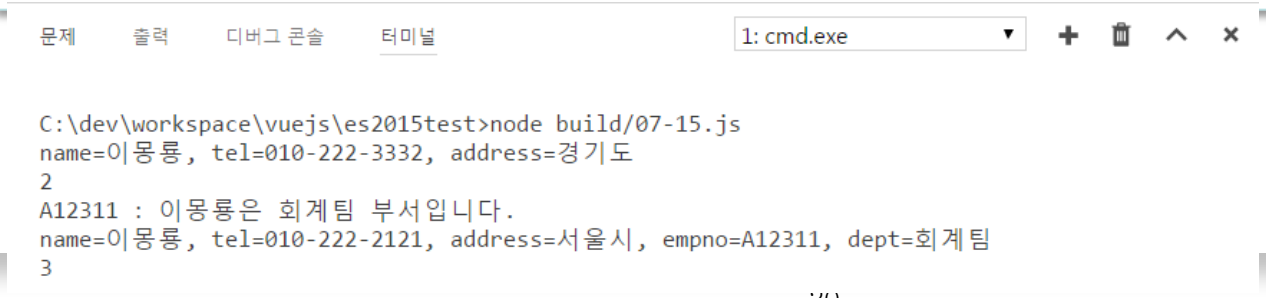
```
class Person {
    constructor(name, tel, address) {
        this.name = name;      this.tel = tel;     this.address = address;
        if (Person.count) { Person.count++; } else { Person.count = 1; }
    }
    static getPersonCount() {
        return Person.count;
    }
    toString() {
        return `name=${this.name}, tel=${this.tel}, address=${this.address}`;
    }
}
var p1 = new Person('이몽룡', '010-222-3332', '경기도');
var p2 = new Person('홍길동', '010-222-3333', '서울');
console.log(p1.toString());
console.log(Person.getPersonCount());
```



클래스

상속 지원

```
//....(기존 코드에 이어서)
class Employee extends Person {
    constructor(name, tel, address, empno, dept) {
        super(name,tel,address);
        this.empno = empno;
        this.dept = dept;
    }
    toString() {
        return super.toString() + `, empno=${this.empno}, dept=${this.dept}`;
    }
    getEmpInfo() {
        return `${this.empno} : ${this.name}은 ${this.dept} 부서입니다.`;
    }
}
let e1 = new Employee("이몽룡", "010-222-2121", "서울시", "A12311", "회계팀");
console.log(e1.getEmpInfo());
console.log(e1.toString());
console.log(Person.getPersonCount());
```



모듈

ES2015에서 공식적으로 모듈 기능 제공

- 전통적인 자바스크립트에서는 모듈의 개념이 희박했음
  - <script> 태그로 참조하는 정도를 의미했음
- 모듈 : 독립성을 가진 재활용 가능한 코드 블록
  - 여러개의 코드 블록을 각각의 파일로 분리시킨 후 필요한 모듈들을 조합해 개발
  - js 코드를 포함하고 있는 파일
  - 코드 블록안에서 import, export 구문을 이용해서 모듈을 가져오거나 내보낼 수 있음
  - 모듈 내부에서 선언된 모든 변수, 함수, 객체, 클래스는 지역적인 것(local)으로 간주
  - 따라서 재사용 가능한 모듈을 만드려면 export 문을 이용해 외부로 공개해야 함

```
export let a= 1000;
export function f1(a) { … }
export { n1, n2 as othername, … }
```

모듈

예제 07-16~18

```
export let var1 = 1000;
export function add(a,b) {
   return a+b;
}
```

```
let var1 = 1000;
function add(a,b) {
   return a+b;
}
export { var1, add };
```

```
import { add, var1 } from './utils/utility1';

console.log(add(4,5));
console.log(var1);
```

```
import { add, var1 as v  } from './utils/utility1';

console.log(add(4,5));
console.log(v);
```

모듈

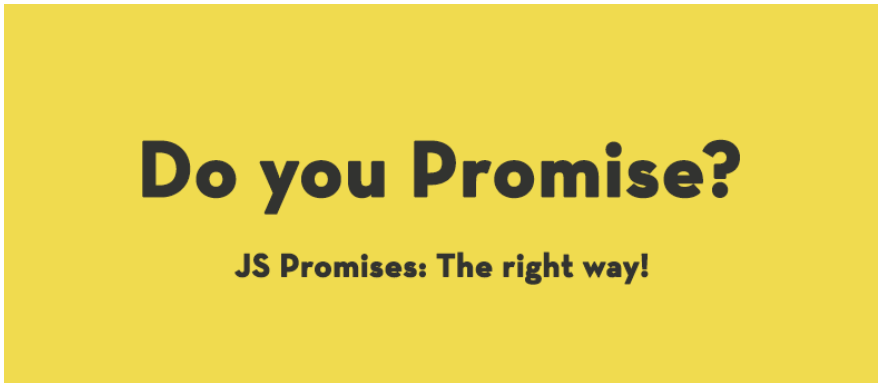
- 단일 값, 함수, 클래스를 export한다면 default 키워드 이용
  - 예제 07-19~20

```
let calc = {
    add(x,y) {
        return x+y;
    },
    multiply(x,y) {
        return x*y;
    }
}

export default calc;
```

```
import calc2 from './utils/utility3';

console.log(calc2.add(4,5));
console.log(calc2.multiply(4,5));
```

Promise

- **AJAX 호출시 비동기 콜백 함수를 주로 사용**
  - 이 방법은 비동기로 처리할 작업이 반복될 때 콜백함수들이 중첩되어 예외처리가 힘들어지고 코드의 복잡도가 증가
- **Promise 객체**
  - 비동기 처리를 좀 더 깔끔하게...
  - axios, fetch, vue-resource, superagent 등의 대표적 HTTP API 들이 대부분 Promise 객체를 사용

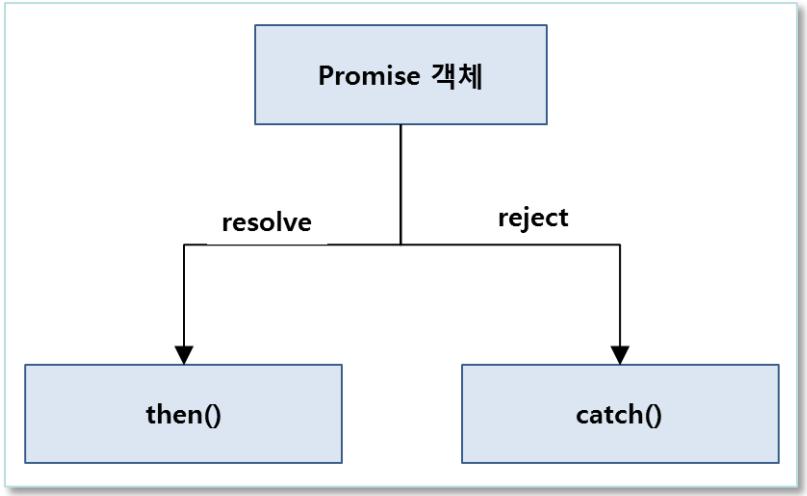


Promise

예제 07-21

- Promise 객체를 생성할 때 전달하는 함수가 비동기로 실행됨
- 비동기로 실행할 코드와 비동기 처리 결과를 받아 실행하는 코드를 분리시킴
  - 첫번째 인자로 전달된 resolve 함수를 호출하면 then 메서드에 등록된 함수가 호출됨.
  - 두번째 인자로 전달된 reject 함수를 호출하면 catch 메서드에 등록된 함수가 호출됨

```
var p = new Promise(function(resolve, reject) {
    setTimeout(function() {
        var num = Math.round(Math.random()*20);
        var isValid = num % 2;
        if (isValid) { resolve(num); }
        else { reject(num); }
    }, 2000);
});
p.then(function(num) {
    console.log("홀수 : " + num);
}).catch(function(num) {
    console.log("짝수 : " + num);
});

console.log("20까지의 난수중 홀수/짝수?");
console.log("결과는 2초후에 나옵니다.!!");
```



Promise

예제 07-22

```
<script>
    let url = "http://sample.bmaster.kro.kr/contacts_long/search/ja";
    fetch(url)
        .then((response)=>response.json())
        .then((json)=> console.log(json))
        .catch((e)=> console.log(e.message));
</script>
```