PWA

- description : PWA
- author : 오션
- email : shlim@repia.com
- lastupdate : 2021-02-01

Ref

- 출처 : “Do it! 프로그레시브 웹앱 만들기 ” , 저자 : 김응석, 2020년 8월, 이지스퍼블리싱

Promise [프로미스]

- 자바스크립트에서 제공하는 비동기를 간편하게 처리할 수 있도록 도와주는 오브젝트이다.
- 정해진 장시간 동안 기능을 수행하고 나서 정상적으로 기능이 수행이 되었다면 성공의 메시지와 함께 처리된 결과값을 전달해준다.
- 기능 수행 시 예상치 못한 문제가 발생하면 에러를 전달해준다.

var 변수 선언

- 함수 스코프(function scope)만 지원한다 -> var 변수는 함수 내부에서만 유효하다.
- 블록 스코프(block scope)는 지원하지 않는다 – > 블록 내부에서var키워드로 변수를 선언해도 블록 외부에서 사용할 수 있다.
- 블록 스코프를 지원하지 않는다는 의미는 함수 외부에서 선언한 변수는 모두 전역 변수처럼 사용된다는 뜻이다.
- var변수는 **중복으로 선언할 수 있다**는 의미는 기존 선언된 변수를 덮어 쓴다는 의미이다
  - 스코프(scope) : 변수의 유효 범위
  - 변수의 유효 범위 : 변수를 사용할 수 있는 범위

mutable[뮤터블, 가변의] vs immutable[이뮤터블, 불변의]

- mutable : 프로그래밍에서 값을 변경할 수 있는 것을 의미 -> let 선언 변수
- immutable : 프로그래밍에서 값을 변경할 수 없는 것을 의미 -> const 선언 변수

let 사용법

```
{
    let 변수명;             // 변수 선언
    let 변수명 = 값;       // 변수 선언과 동시에 값 할당
}
```

const 사용법

```
{
    const 변수명 = 값;        // 변수 선언과 동시에 반드시 값을 할당해야 함
}
```

블록 스코프(block scope) 예시 - let과 const로 선언한 변수의 사용법

```
{
    let apple = 3;
    console.log(apple);        // 3 출력
    apple = 5;                 // let  선언 변수는 재할당 허용
    console.log(apple);        // 5 출력
    let apple = 7;             // 같은 이름으로 중복 선언을 허용하지 않으므로 오류!

    const car;                 // 선언과 동시에 값을 할당하지 않으므로 오류!
    const carname = "하이브리드카";
    console.log(carname);      // '하이브리드카' 출력
    carname = "디젤카"        //  const  선언 변수는 불변이므로 오류;
}
console.log(apple);          // 블록 밖이므로 오류!
console.log(carname);        // 블록 밖이므로 오류!
```

모던 자바스크립트에서 변수 사용 3원칙

- let과 const 키워드로 선언한 변수는 유효 범위가 블록 스코프로 제한된다

1. const를 가장 먼저 사용한다. 특히 변숫값을 변경할 필요가 없을 경우 사용한다
2. 만약 중간에 변숫값을 변경해야 한다면 const 대신에 let을 사용한다.
3. var는 ES6+에서는 사용하지 않는다. 다만 전역 변수가 꼭 필요할 때 최상위 수준에서만 예외로 사용한다.

템플릿 리터럴 [Template Literal]

- ES6부터는 역따옴표(백틱 ``)을 사용할 수 있다. 이처럼 역따옴표로 묶은 문자열을 '템플릿 리터럴'이라고 한다.
- 템플릿 리터럴을 사용하면 문자열 안에 달러 기호($)와 중괄호({ })를 사용해서 표현식을 넣을 수 있으며 여러 줄로 만들어 사용하기 편리하다.
- 이전처럼 문자열을 + 연산자로 연결하지 않아도 표현식을 문자열과 함께 조합해서 사용할 수 있다.
- 템플릿 리터럴 예시

```
console.log('사과:' + apple + ', 자동차:' + carname);       // 이전 스타일
console.log(`사과: ${apple}, 자동차: ${carname}`);         // ES6 스타일(템플릿 리터럴)
```

화살표 함수

- 화살표 함수(arrow function)는 ES6 문법에서 함수를 표현하는 새로운 방식이며 가장 효과적으로 변경된 문법이며, 모던 자바스크립트 프로그래밍에서 가장 많이 사용하는 방식이다
- ⇒ 는 두꺼운 화살이라는 의미로 '팻 애로우(fat arrow)'라고 한다

전통적인 함수 선언 방식

```
const fnPlusNumbers = function (pNum1, pNum2) {
   return pNum1 + pNum2;
}
```

화살표 함수로 구현

```
const fnPlusNumbers = (pNum1, pNum2) => {
   return pNum1 + pNum2;
}
```

return 키워드를 생략한 화살표 함수

```
const fnPlusNumbers = (num1, num2) => num1 + num2;
```

- 함수가 실행되는 문장(statement)가 단 한 줄이므로 블록을 나타내는 중괄호도 생략할 수 있다.
- 화살표 함수는 return키워드를 생략해도 자동으로 값을 반환한다. 이것을 '암묵적 반환'이라고 한다. 화살표 뒤에 있는 num1 + num2의 결괏값이 자동으로 반환된다

- 매개변수가 1개일 때 소괄호 ()를 생략할 수 있고, 매개변수가 없을 때는 반드시 소괄호를 사용해야 한다.

- 매개변수가 1개일 때 소괄호 생략 가능

```
const fnPlusNumbers4 = pNum => pNum + 1;
```

- 매개변수가 없을 때는 반드시 소괄호 사용

```
const fnSayHello = () => '안녕하세요!';
```

동기 처리 방식과 비동기 처리 방식 비교

구분	동기 처리 방식	비동기 처리 방식
실행 순서	모든 코드가 위에서 아래 순서로 실행	코드들이 독립적으로 실행
실행 대기	명령을 요청한 후 결과가 나올 때까지 계속 기다림	명령을 요청한 후 결과가 나올 때까지 기다리지 않고 다음 작업을 연이어 수행
	반환 결과를 계속 기다리므로 시간 지연이 큼	결과가 완료되면 다시 돌아와 필요한 것을 마무리하는 방식

콜백 함수

- 자바스크립트는 비동기 처리 방식을 구현하기 위해 콜백 함수(callback function)를 사용한다.
- 함수는 보통 매개변수로 값을 전달받는데 자바스크립트에서 함수는 객체(Object)로 취급하므로 함수의 매개변수로 **함수**를 전달받을 수 있다.
- 어떤 함수의 매개변수로 전달된 함수를 콜백 함수(callback function)라고 부른다.
- 콜백 함수는 다른 함수와 상관없이 독립적으로 실행되므로 비동기 처리를 구현할 때 사용한다.
- 매개변수로 전달받은 함수를 콜백 함수로 이용할 때는 관용적으로 매개변수 이름에 'callback'이라는 단어를 함께 표기한다.

- 콜백 함수를 이용한 비동기 처리

```
function fnFunctionA(pNumA) {
   console.log(pNumA);
}
function fnFunctionB(pNumB, fnCallback) {
   fnCallback(pNumB);     // 매개변수로 받은 함수를 실행하여 비동기 처리의 시작 담당
}

fnFunctionB("콜백 함수 실행!", fnFunctionA);
```

- 콜백 함수 구조를 단순화한 예
- fnFunctionB() 함수를 실행할 때 두 번째 매개변수에 콜백 함수 선언과 정의를 함께 작성한 방식이

며, 실무에서 이 방식을 선호한다.

```
function fnFunctionB(pNumB, fnCallback) {
   fnCallback(pNumB);
}

// 매개변수 전달 시 함수 선언과 정의를 함께 작성함: 비동기 처리의 완료 담당
fnFunctionB("콜백 함수 실행!", function fnFunctionA(pNumA) {
   console.log(pNumA);
});
```

프로퍼티 접근

- 프로퍼티에 접근하는 방법은 다음과 같이 두 가지이다.
  - 마침표 프로터피 접근 연산자(**.**)를 사용하는 **마침표 표기법**$^{\textbf{dot notation}}$
  - 대괄호 프로퍼티 접근 연산자(**[...]**)를 사용하는 **대괄호 표기법<sp>bracket notation</sup>**
- 프로퍼티 키가 식별자 네이밍 규칙을 준수하는 이름, 즉 자바스크립트에서 사용 가능한 유효한 이름이면 마침표 표기법과 대괄호 표기법을 모두 사용할 수 있다.
- 마침표 프로퍼티 접근 연산자 또는 대괄호 프로퍼티 접근 연산자의 좌측에는 객체로 평가되는 표현식을 기술한다. 마침표 프로퍼티 접근 연산자의 우측 또는 대괄호 프로퍼티 연산자의 내부에는 프로퍼티 키를 지정한다.

```
var person = {
  name: 'Lee'
};

// 마침표 표기법에 의한 프로퍼티 접근
console.log(person.name);    // Lee

// 대괄호 표기법에 의한 프로퍼티 접근
console.log(person['name']);    // Lee
```

- 대괄호 표기법을 사용하는 경우, **대괄호 프로퍼티 접근 연산자 내부에 지정하는 프로퍼티 키는 반드시 따옴표로 감싼 문자열**이어야 한다. 대괄호 프로퍼티 접근 연산자 내에 따옴표로 감싸지 않은 이름을 프로퍼티 키로 사용하면 자바스크립트 엔진은 식별자로 해석한다.

```
var person = {
   name: 'Lee'
};

console.log(person[name]);    // ReferenceError: name is not defined
```

* 위 예제에서 ReferenceError가 발생한 이유는 대괄호 연산자 내의 따옴표로 감싸지 않은 이름, 즉 식별자 name을 평가하기 위해 선언된 name을 찾았지만 찾지 못했기 때문이다. * **객체에 존재하지 않는 프로퍼티에 접근하면 undefined를 반환한다.** 이때 ReferenceError가 발생하지 않는 데 주의하자.

```
var person = {
   name: 'Lee'
};

console.log(person.age);    // undefined
* 프로퍼티 키가 식별자 네이밍 규칙을 준수하지 않는 이름, 즉 자바스크립트에서 사용 가능한 유효한
이름이 아니면 반드시 대괄호 표기법을 사용해야 한다. 단, 프로퍼티 키가 숫자로 이뤄진 문자열인 경
우 따옴표를 생략할 수 있다. 그 외의 경우 대괄호 내에 들어가는 프로퍼티 키는 반드시 따옴표로 감싼
문자열이어야 한다는 점을 잊지 말자.
```

ES11, ES6 Syntax Summery

- 출처 : 드림코딩 by 엘리, 자바스크립트 최신 문법 (ES6, ES11) | 모르면 후회하는 최신 문법과 사용법 정리

- Optional Chaining (ES11)
    - Optional Chaining

- Nullish Coalescing Operator (ES11)
    - Nullish Coalescing Operator

- Shorthand property names (ES6)
    - 객체 초기자

- Destructuring Assignment (ES6)
    - 구조 분해 할당

- Spread Syntax (ES6)
    - 전개 구문

- Default parameter (Es6)
    - 기본값 매개변수

- Ternary Operator (ES6)
    - 삼항 조건 연산자

- Template Literals (ES6)

    - 템플릿 리터럴

오션, pwa