# 서버 프로그램을 위한 자바스크립트

| | |
|---|---|
| 생성일 | @2024년 11월 4일 오후 12:25 |
| 태그 | |

▼ 변수 선언자 이해하기

▼ var 선언자

- 특징

1. 함수 레벨 스코프

▼ 스코프(Scope, 유효범위)

- 정의
  - 참조 대상 식별자(identifier)를 찾아내기 위한 규칙
- 구분
  - 전역 스코프
  - 지역 스코프
- 종류
  - 블록 레벨 스코프(block-level scope)
    - 코드 블록({...})내에서 유효
    - C 계열 프로그래밍 언어에서 적용
  - 함수 레벨 스코프(Function-level scope)
    - 함수 코드 블록 내에서 선언된 변수는 함수 코드 블록 내에서만 유효
    - 함수 밖 선언된 변수의 경우 코드 블록 내 선언에도 모두 전역 스코프

- 전역 함수 밖에서 생성한 변수는 모두 전역 변수로
  코드 내 전역 변수를 남발할 가능성이 높음
- for 문의 변수 선언문에서 선언한 변수를 외부에서 참조할 수 있음

2. var 키워드 생략 허용
   - 암묵적으로 전역 변수를 생성할 가능성이 높음
3. 변수 중복 선언 허용
   - 의도하지 않은 변수의 값 변경이 일어날 수 있음
4. 변수 호이스팅

▼ 호이스팅

- 정의
  - 모든 선언문이 해당 스코프의 선두로 옮겨진 것처럼 동작하는 특성
- 과정
  1. 선언 단계(Declaration phase)

     변수 객체(Variable Object)에 변수를 등록
  2. 초기화 단계(Initialization phase)

     변수 객체(Variable Object)에 등록된 변수를 메모리에 할당(undefined)
  3. 할당 단계(Assignment phase)

     undefined로 초기화된 변수에 실제 값을 할당
- 특징
  - 모든 선언문(var, let, const, function 등)이 선언되기 이전에 참조 가능

                - 변수를 선언하기 이전에 참조할 수 있음
        - 장점
            - 동일한 이름의 변수를 중복하더라도 에러 발생되지 않음
        - 단점
            - 외부 라이브러리를 사용하거나 협업하는 경우
                - 동일한 변수명 사용으로 인한 에러 발생 → 추적이 어려움
- var 선언자의 문제점을 해소하기 위해 ES6에서 추가된 선언자
    - let 선언자
        1. 블록 레벨 스코프
        2. 변수 중복 선언 불가
        3. 호이스팅
            - 특징
                - 선언 단계와 초기화 단계가 분리되어 진행

                ▼ 과정

                1. 선언 단계(Declaration phase)

                   변수 객체(Variable Object)에 변수를 등록
                2. 일시적 사각지대(Temporal Dead Zone)
                    - 선언 단계와 초기화 단계 사이 구간
                    - 실제 변수 선언문을 만나기 전까지 메모리 공간 미할당
                    - 변수에 접근하려고 하면 참조 에러(ReferenceError) 발생
                3. 초기화 단계(Initialization phase) → 실제 변수를 선언한 부분에서 실행

                   변수 객체(Variable Object)에 등록된 변수를 메모리에 할당(undefined)
                4. 할당 단계(Assignment phase)

                   undefined로 초기화된 변수에 실제 값을 할당
            - 동일한 변수 재사용으로 인한 오류를 방지하기 위해 기본 선언자로 사용
    - const 선언자
        1. 블록 레벨 스코프
        2. 변수 선언과 동시에 초기화 → 재할당 불가
        3. 상수
            - 객체을 할당하는 경우
                - 참조 타입인 객체의 경우 객체는 변경할 수 없지만
                  내부 프로퍼티는 값을 변경할 수 있음
            - 상수를 사용하거나 특정 포맷을 유지해야 하는 경우 사용

▼ 화살표 함수(Arrow Function)

- 활용
    - function 키워드 대신 화살표(=>)를 사용하여 함수를 보다 간략한 방법으로 선언

▼ 문법

- 매개변수 표현

```
// 매개변수가 없을 경우
    () => { ... }

// 매개변수가 한 개인 경우 (소괄호를 생략가능)
```

```
      x => { ... }

// 매개변수가 여러 개인 경우
(x, y) => { ... }
```

- 함수 몸체 표현

```
// Default
(x, y) => { return x * y }
(x, y) => {
                        let result = x * y;
                        return result + 10;
                    }

// 실행코드가 한줄이라면 중괄호를 생략, 암묵적으로 결과를 return함
(x, y) => x * y
```

- this
  - 일반함수
    - 함수 호출 방식에 따라 this에 바인딩될 대상이 동적으로 결정
  - 화살표 함수
    - 전역 객체 window
  - 함수 내부에서 this를 사용하는 경우 화살표 함수는 사용불가

▼ Array 내장 함수

▼ sort()

- 기능
  - 배열의 데이터를 정렬
  - 기본적으로 모든 데이터를 문자로 인식하여 유니코드를 기반으로 오름차순 정렬
    - 숫자의 경우 별도의 정렬 함수를 정의해서 사용
- 문법

```
let array = [];

// 배열의 값이 문자 타입인 경우
array.sort();        // 오름차순
array.reverse();     // 내림차순

// 배열의 값이 문자 타입이 아닌 경우
// return 값이 양수일 경우 a(첫번째 매개변수)가 크다고 인식
// return 값이 음수일 경우 b(두번째 매개변수)가 크다고 인식
array.sort(function(a, b){
    return a - b;
});
```

▼ filter()

- 기능
  - 전체 데이터 중 특정 조건을 만족하는 배열의 요소만을 찾아 새로운 배열로 반환
  - 원본 배열은 변경되지 않음
- 문법

```
let array = [];

array.filter(callbackFn(element[,index[,array]])[, thisArg]);
```

▼ map()

- 기능
  - 전체 데이터 중 각 요소에 대해 콜백 함수의 반환값으로 새로운 배열을 생성해 반환
  - 원본 배열은 변경되지 않음
- 문법

```
let array = [];

array.map(callbackFn(element[,index[,array]])[, thisArg]);
```

- 용도
  1. 전체 데이터의 각 요소에 대해 데이터를 추가해야 하는 경우
  2. 전체 데이터의 각 요소에 대해 필요한 데이터를 선별해야 하는 경우

▼ reduce()

- 기능
  - 배열에 담긴 데이터를 순회하며 콜백 함수의 실행 값을 누적한 결과 값을 반환
- 문법

```
let array = [];

array.reduce(callback[, initialValue]);

function callback (accumulator,currentValue[,currentIndex[,array]]){
    accumulator  : 누적값
    currentValue : 현재 배열의 요소
    currentIndex : 현재 배열의 인덱스
    array        : 현재 사용되는 배열
}
```

▼ 템플릿 리터럴(Template Literals)

- ES6에서 도입된 새로운 문자열 표기법
- 특징
  - ` (백틱, backtick) 문자 사용
    - 내부에서 홀따옴표와 백따옴표를 단순 특수문자로 혼용해서 사용
  - 문자열 인터폴레이션(String Interpolation)
    - 문자열 내부에 + 없이 변수 사용을 허용 : ${ ... }

▼ 펼침 연산자(Spread Operator)

- 정의
  - 이터레이션(iteration) 형태의 데이터를 개별 요소로 분리
  - 대표적으로 배열, 문자열이 있으며
    이터레이션 프로토콜(iteration protocol)을 준수한 객체도 대상이 될 수 있음

  ▼ 이터레이션 프로토콜(iteration protocol)

  - 데이터 컬렉션을 순회하기 위해 미리 약속된 규칙
  - 종류

1. 이터러블 프로토콜(iterable protocol)
    - 이터러블(iterable)
        - 해당 프로포콜을 준수한 객체
        - Symbol.iterator 메소드를 구현하거나 프로토타입 체인에 의해 상속
2. 이터레이터 프로토콜(iterator protocol)
    - 이터레이터(iterator)
        - 해당 프로토콜을 준수한 객체
        - next 메소드를 소유하고 next()를 호출하면 이터러블을 순회하며 이터레이터 리절트 객체( value, done 프로퍼티 포함)를 반환

- 문법

```
let array = [];
let str = "";

let newAray = [...array, ...str];
```

▼ 구조 분해 할당(Destructuring assignment)

- 객체나 배열의 내부 요소를 개별 변수로 분해
- Object Destructuring
    - 프로퍼티 키를 기준으로 객체로부터 추출하여 변수 리스트에 할당
    - 문법

```
let obj = {
    id : 100,
  name : 'Hong',
  birth : '1999-12-25'
}

let {id, name, birth} = obj;
```

- Array Destructuring
    - 배열의 인덱스를 기준으로 배열로부터 요소를 추출하여 변수에 할당
    - 문법

```
let array = [1, 2, 3];

// 1
let [a, b, c] = array;

// 2
let x, y, z;
[x,y,z] = array;
```

▼ 매개변수 기본값(Default Function Parameter)

- 정의
    - 함수를 호출할 때 해당 매개변수가 전달되지 않을 경우 기본값을 설정
- 문법

```
function hello ( msg = 'World' ){
    return 'Hello, ' + msg;
}

console.log(hello());   // "Hello, World"
```

▼ 나머지 매개변수(Rest Parameter)

- 정의
  - 이름 앞에 세개의 점 `...`을 붙여서 정의한 매개변수
  - 함수에 전달된 인수들의 목록을 배열로 전달
- 문법

```
// 더하는 수의 제한이 없는 더하기 계산
function sum(x, y, ...args){
    let result = x + y ;
    for(let num of args){
        result += num;
    }
    return result;
}
```

▼ Promise

- 정의
  - 비동기 처리에 사용하는 객체
  - 주로 파일 쓰기, 데이터베이스 CRUD 처리, AJAX 통신 등에 사용
- 처리 상태

| 상태 | 의미 | 구현 |
|---|---|---|
| pending | 아직 수행되지 않은 상태 | 함수가 아직 호출되지 않은 상태 |
| **fulfilled** | 수행된 상태 (성공) | resolve 함수가 호출된 상태 |
| **rejected** | 수행된 상태 (실패) | reject 함수가 호출된 상태 |
| settled | 수행된 상태 (성공 또는 실패) | 함수(resolve or reject)가 호출된 상태 |

- 문법

```
const promise = new Promise((resolve, reject) => {
    if(true){
        /* 처리 성공 */
        resolve("결과 데이터");
    }else{
        /* 처리 실패 */
        reject(new Error("에러 메세지"));
    }
}

promise
.then((success, fail)) => {
    /* 비동기 처리 결과*/
    /*
            success (첫번째 매개변수) : 성공(fulfilled, resolve 함수 호출상태) 시 호출
            fail    (두번째 매개변수) : 실패(rejected, reject 함수 호출상태) 시 호출
    */
})
.catch(error => {
```

```
    /* 예외(비동기 처리에서 발생한 에러와 then 메소드에서 발생한 에러)가 발생 */
})
.finally(()=>{
    /* 처리결과와 무관하게 항상 실행해야하는 코드 */
})
```

▼ Async/Await

- 정의
  - 비동기 처리에 사용하는 문법
  - 동일한 스코프에서 결과 값을 관리 → 효율적으로 프로그램 구현
- 문법

```
// 함수 내부에 await가 존재한는 경우 function 앞에 반드시 async
async function getJSONData(){
    const result = await webServerConnect();
    console.log(result);
}

function webServerConnect(){
    return fetch('https://jsonplaceholder.typicode.com/posts/1',{
    headers : {
        'Cache-Control': 'no-cache'
    }
  })
  .then(response => response.json());
}

getJSONData();
```

▼ 클래스(Class)

- 기본 문법

```
class 클래스명 {
    constructor(매개변수){ // class로 정의된 객체를 생성하고 초기화, 단 하나만 존재
        // 초기화 코드 작성, 필요에 따라 this를 사용하여 내부 변수 선언
    }

    // 외부에서 클래스의 변수에 접근하기 위한 set, get 선언
    // set만 선언할 경우 변수 값 반환 불가
    // get만 선언할 경우 변수에 값 할당 불가

    set 변수명(매개변수){
        // constructor에 this로 선언한 변수에 값 할당
    }

    get 변수명(){
        // constructor에 this로 선언한 변수의 값을 반환
        return this.변수명;
    }
}
```

- 상속 문법

```
class 자식클래스명 extends 부모클래스명 {
    constructor(매개변수){
```

```
        super(매개변수);      // 부모클래스가 가진 constructor에 매개변수를 전달할 경우
    }
}
```

▼ 정규 표현식(Regular Expression, 정규식)

- 정규식 만들기
  1. 정규식 리터럴

     ```
     const regexp = /World/;
     ```

  2. RegExp 객체

     ```
     const regexp = new RegExp("World");
     ```

- 정규식 함수

| 종류 | 객체 | 설명 |
| --- | --- | --- |
| `exec()` | RegExp | 문자열에서 일치하는 부분을 탐색<br>일치 정보를 나타내는 배열, 또는 일치가 없는 경우<br>`null`을 반환 |
| `test()` | RegExp | 문자열에 일치하는 부분이 있는지 확인 (`true` 또는 `false`) |
| `match()` | String | 캡처 그룹을 포함해서 모든 일치하는 부분를 담은 배열을 반환<br>일치가 없으면<br>`null`을 반환 |
| `search()` | RegExp | 문자열에서 일치하는 부분을 탐색<br>일치하는 부분의 인덱스, 또는 일치가 없는 경우<br>`-1`을 반환 |
| `replace()` | String | 문자열에서 일치하는 부분을 탐색하고, 그 부분을 대체 문자열로 바꿈 |
| `split()` | String | 정규 표현식 또는 문자열 리터럴을 사용해서<br>문자열을 부분 문자열의 배열로 반환 |

- 정규식 플래그
  - 검색 방법에 대한 조건

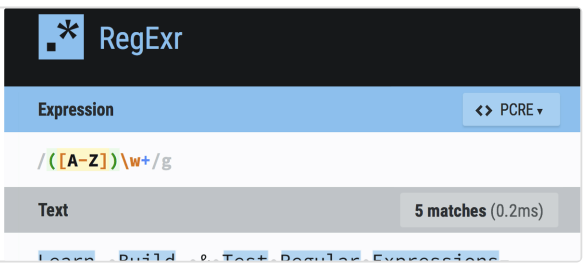
| 플래그 | 설명 |
| --- | --- |
| g(global) | 전역 검색 - 대응되는 문자 전부 검색 |
| i(ignoring case) | 대소문자 구분 없는 검색 |
| m(multiline) | 다중 행 검색 |

- 정규식 테스트 사이트

RegExr: Learn, Build, & Test RegEx

RegExr is an online tool to learn, build, & test Regular Expressions (RegEx / RegExp).

https://regexr.com/




  - 사이트 설명(한국 블로그)

86. 정규표현식 테스트 사이트 : regexr.com

0. 차례 1. 들어가기 2. regexr.com에서 정규 표현식 테스트하기 3. 마무리... 4. 참고자료(Reference) 1....

https://m.blog.naver.com/pmw9440/221847884961