Chap21 빌트인 객체

21-1 자바스크립트 객체분류

▼ 표준 빌트인 객체

ECMAScript 사양에 정의된 객체를 말하며 애플리케이션 전역의 공통 기능을 제공한다. 표준 빌트인 객체는 ECMAScript 사양에 정의된 객체이므로 자바스크립트 실행환경과 관계없이 언제나 사용할 수 있다. 표준 빌트인 객체는 전역 객체의 프로퍼티로서 제공된다.

▼ 호스트 객체

ECMAScript 사양에 정의되어 있지 않지만 자바스크립트 환경(브라우저 또는 Node.js)에서 추가로 제공하는 객체를 말한다.

브라우저 환경에서는 DOM,BOM,Canvas,CMLTttpRequest,fetch,requestAnimationFrame,SVG, Web Storage, WebComponent, Web Worker와 같은 클라이언트 사이드 Web API를 호스트 객체로 제공하고, Node.js환경에서는 Node.js 고유의 API를 호스트 객체로 제고한다.

▼ 사용자 정의 객체

표준 빌트인 객체와 호스트 객체처럼 기본 제공되는 객체가 아닌 사용자가 직접 정의한 객체를 말한다.

21-2 표준 빌트인 객체

▼ 표준 빌트인 객체 종류

Object, String, Number, Boolean, Symbol, Date, Math, RegExp, Array, Map/Set, WeakMap, WeakSet, Function, Promise, Reflect, Proxy, JSON, Error

▼ 인스턴스 생성 가능한 생성자 함수 객체

Object, String, Number, Boolean, Symbol, Date, RegExp, Array, Map/Set, WeakMap, WeakSet, Function, Promise, Proxy, Error

▼ 특징

프로토 타입 메서드와 정적 메서드를 제공한다.(생성자 함수 객체가 아닌 표준 빌트인 객체는 정적 메서드만 제공한다.)

21-3 원시 값과 래퍼 객체

▼ 래퍼 객체

- 문자열, 숫자, 불리언 원시 값에 대해 객체처럼 접근하면 생성되는 임시 객체
- 원시 타입은 자바스크립트의 기본 데이터 타입으로, 문자열(`String`), 숫자(`Number`), 불리언(`Boolean`), 심볼(`Symbol`), `null`, `undefined`가 있습니다. 이 중에서 `null`과 `undefined`를 제외한 나머지 원시 타입은 래퍼 객체가 존재합니다.

```
const str='hello';
console.log(str.length);//5
console.log(str.toUpperCase());//HELLO
```

위 예시처럼 원시 값인데도 불구하고 객체처럼 프로퍼티와 메소드를 사용할 수 있다.

▼ Symbol도 래퍼객체?

yes

21-4 전역 객체

▼ 빌트인 전역 프로퍼티

전역 객체의 프로퍼티를 의미한다. 주로 전역에서 사용하는 값을 제공

- Infinity : 무한대를 나타내는 숫자값 Infinity를 갖는 프로퍼티
- NaN : 숫자가 아님을 나타내는 숫자값 NaN을 갖는 프로퍼티. Number.NaN프로퍼티와 같다.
- undefined : 원시 타입 undefined를 값으로 갖는 프로퍼티

▼ 빌트인 전역 함수

애플레케이션 전역에서 호출할 수 있는 빌트인 함수로서 전역 객체으 메서드

▼ eval

eval함수는 자바스크립트 코드를 나타내는 문자열을 인수로 전달받아, 전달받은 문자열 코드가 표현식 이라면 eval함수는 문자열 코드를 런타임에 평가하여 값을 생성하고,
전달받은 인수가 표현식이 아닌 문이라면 eval함수는 문자열 코드를 런타임에 실행한다.

```
// 표현식인 문
eval('1 + 2;'); // -> 3

// 표현식이 아닌 문
eval('var x = 5;'); // -> undefined

// eval 함수에 의해 런타임에 변수 선언문이 실행되어 x 변수가 선언도
console.log(x); // 5

// 객체 리터럴은 반드시 괄호로 둘러싼다.
const o = eval('({ a: 1 })');
console.log(o); // {a: 1}

// 함수 리터럴은 반드시 괄호로 둘러싼다.
const f = eval('(function() { return 1; })');
console.log(f()); // 1
```

여러개의 문으로 이루어져 있다면 모든 문을 실행한 다음, 마지막 결과값을 반환한다.

```
console.log(eval('1 + 2; 3 + 4;')); // 7
```

`eval` 함수는 자신이 호출된 위치에 해당하는 기존의 스코프를 런타임에 동적으로 수정한다.

```
eval함수는 자신이 호출된 위치에 해당하는 기존의 스코프를 런타임에 동
```

`strict mode`에서는 `eval`함수는 기존의 스코프를 수정하지 않고 `eval`함수 자신의 자체적인 스코프를 생성한다.

```
const x = 1;

function foo() {
  'use strict';

  // strict mode에서 eval 함수는 기존의 스코프를 수정하지 않고
  eval('var x = 2; console.log(x);'); // 2
  console.log(x); // 1
}

foo();
console.log(x); // 1
```

인수로 전달받은 문자열 코드가 `let`, `const` 키워드를 사요한 변수 선언문이라면 암묵적으로 `strict mode`가 적용된다.

```
const x = 1;

function foo() {
  eval('var x = 2; console.log(x);'); // 2
  // let, const 키워드를 사용한 변수 선언문은 strict mode가 적
  eval('const x = 3; console.log(x);'); // 3
  console.log(x); // 2
}

foo();
console.log(x); // 1
```

▼ eva 함수의 단점

보안에 매우 취약하다.

또한, 최적화가 수행되지 않아 일반적인 코드 실행에 비해 느리다.

따라서 `eval` **함수의 사용은 금지해야 한다.**

▼ inFinite

전달받은 인수가 정상적인 유한수 인지 검사하여 유한수이면 `true`, 무한수이면 `false`를 반환한다.

▼ isNaN

전달받은 인수가 `NaN`인지 검사하여 그 결과를 불리언 타입으로 반환한다. 전달받은 인수의 타입이 숫자가 아닌경우 숫자로 타입을 변환한 후 검사를 수행한다.

▼ parseFloat

전달받은 문자열 인수를 부동 소수점 숫자(`floating point number`)즉, 실수로 해석하여 반환한다.

```
// 문자열을 실수로 해석하여 반환한다.
parseFloat('3.14');  // -> 3.14
parseFloat('10.00'); // -> 10

// 공백으로 구분된 문자열은 첫 번째 문자열만 변환한다.
parseFloat('34 45 66'); // -> 34
parseFloat('40 years'); // -> 40

// 첫 번째 문자열을 숫자로 변환할 수 없다면 NaN을 반환한다.
parseFloat('He was 40'); // -> NaN

// 앞뒤 공백은 무시된다.
parseFloat(' 60 '); // -> 60
```

▼ parseInt

- 전달받은 문자열 인수를 정수로 해석하여 반환한다. 두 번째 인수로 진법을 나타내는 기수(2~36)을 전달할 수 있다
- 문자열이 16진수 리터럴(`0x`또는 `0X`로 시작)이면 16진수로 해석하여 10진수 정수로 반환한다.
- 2진수, 8진수 리터럴은 안된다. 0으로 시작하는 숫자로 인식해 버린다.

```
// 2진수 리터럴(0b로 시작)은 제대로 해석하지 못한다. 0 이후가 무시된다.
parseInt('0b10'); // -> 0
// 8진수 리터럴(ES6에서 도입. 0o로 시작)은 제대로 해석하지 못한다. 0 이후가 무시된다.
parseInt('0o10'); // -> 0
```

참고

```
// 'A'는 10진수로 해석할 수 없다.
parseInt('A0'); // -> NaN
// '2'는 2진수로 해석할 수 없다.
parseInt('20', 2); // -> NaN

// 10진수로 해석할 수 없는 'A' 이후의 문자는 모두 무시된다.
parseInt('1A0'); // -> 1
// 2진수로 해석할 수 없는 '2' 이후의 문자는 모두 무시된다.
parseInt('102', 2); // -> 2
// 8진수로 해석할 수 없는 '8' 이후의 문자는 모두 무시된다.
parseInt('58', 8); // -> 5
// 16진수로 해석할 수 없는 'G' 이후의 문자는 모두 무시된다.
parseInt('FG', 16); // -> 15

// 공백으로 구분된 문자열은 첫 번째 문자열만 변환한다.
parseInt('34 45 66'); // -> 34
parseInt('40 years'); // -> 40
// 첫 번째 문자열을 숫자로 변환할 수 없다면 NaN을 반환한다.
parseInt('He was 40'); // -> NaN
// 앞뒤 공백은 무시된다.
parseInt(' 60 '); // -> 60
```

▼ `Number.prototype.toString`메서드

기수를 지정하여 10진수 숫자를 문자열로 변환하여 반환

▼ encodeURI / decodeURI

▼ 인코딩

URI 문자들으 이스케이프 처리하는 것

▼ 이스케이프 처리

네트워크를 통해 정보를 공유할 때 어떤 시스템에서도 읽을수 있는 아스키 문자셋으로 변환하는 것

▼ encodeURI / decodeURI

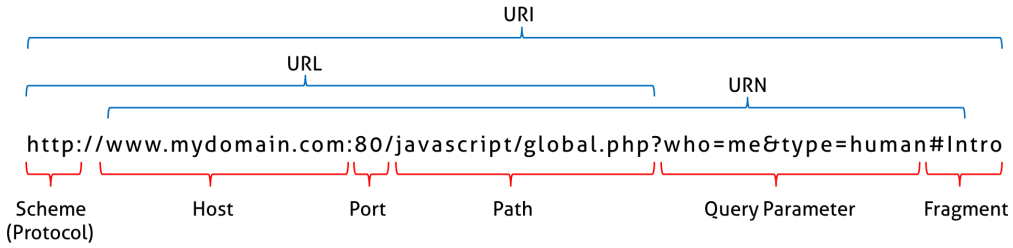
decodeURI함수는 인코딩 된 URI를 인수로 전달받아 이스케이프 처리 이전으로 디코딩한다.

```
const uri = 'http://example.com?name=이웅모&job=progra

// encodeURI 함수는 완전한 URI를 전달받아 이스케이프 처리를 수
const enc = encodeURI(uri);
console.log(enc);
// http://example.com?name=%EC%9D%B4%EC%9B%85%EB%AA%A

// decodeURI 함수는 인코딩된 완전한 URI를 전달받아 이스케이프
const dec = decodeURI(enc);
console.log(dec);
// http://example.com?name=이웅모&job=programmer&teach
```

▼ URI & URL & URN



URI란?

- URI는 Uniform Resource Identifier, 통합 자원 식별자의 줄임말이다.
- 즉, URI는 인터넷의 자원을 식별할 수 있는 문자열을 의미한다.
- URI의 하위 개념으로 URL과 URN이 있다.
- URI 중 URL이라는, URN이라는 하위 개념을 만들어서 특별히 어떤 표준을 지켜서 자원을 식별하는 것이다.
- 결론은 URI라는 개념은 어떤 형식이 있다기 보다는 특정 자원을 식별하는 문자열을 의미한다. 그래서 URL이 아니고 URN도 아니면 그냥 URI가 되는 것이다.

URL이란?

- URL은 Uniform Resource Locator의 줄임말이다.
- URL은 네트워크 상에서 리소스(웹 페이지, 이미지, 동영상 등의 파일) 위치한 정보를 나타낸다.

- URL은 HTTP 프로토콜 뿐만아니라 FTP, SMTP 등 다른 프로토콜에서도 사용할 수 있다.
- URL은 웹 상의 주소를 나타내는 문자열이기 때문에 더 효율적으로 리소스에 접근하기 위해 클린한 URL 작성을 위한 방법론이다.

URN이란?

- URN은 Uniform Resource Name의 줄임말이다.
- URN은 URI의 표준 포맷 중 하나로, 이름으로 리소스를 특정하는 URI이다.
- http와 같은 프로토콜을 제외하고 리소스의 name을 가리키는데 사용된다.
- URN에는 리소스 접근방법과, 웹 상의 위치가 표기되지 않는다.
- 실제 자원을 찾기 위해서는 URN을 URL로 변환하여 이용한다.

▼ encodeURIComponent / decodeURIComponent

▼ encodeURIComponent

URI구성 요소를 인수로 전달받아 인코딩한다.

encodeURIComponent 함수는 인수로 전달된 문자열을 URI의 구성요소인 쿼리 스트링의 일부로 간주한다. 따라서 쿼리 스트링 구분자로 사용되는 =,?,& 까지 인코딩한다.

```
// URI의 쿼리 스트링
const uriComp = 'name=이웅모&job=programmer&teacher';

// encodeURIComponent 함수는 인수로 전달받은 문자열을 URI의
// 따라서 쿼리 스트링 구분자로 사용되는 =, ?, &까지 인코딩한다
let enc = encodeURIComponent(uriComp);
console.log(enc);
// name%3D%EC%9D%B4%EC%9B%85%EB%AA%A8%26job%3Dprogra

let dec = decodeURIComponent(enc);
console.log(dec);
// 이웅모&job=programmer&teacher
```

```
// encodeURI 함수는 인수로 전달받은 문자열을 완전한 URI로 간주
// 따라서 쿼리 스트링 구분자로 사용되는 =, ?, &를 인코딩하지 않
enc = encodeURI(uriComp);
console.log(enc);
// name=%EC%9D%B4%EC%9B%85%EB%AA%A8&job=programmer&t

dec = decodeURI(enc);
console.log(dec);
// name=이웅모&job=programmer&teacher
```

▼ 암묵적 전역

선언하지 않은 변수가 전역 객체의 프로퍼티가 되는 현상