04

Javascript OOP

JavaScript

인터뷰에서 이 부분에 대한 질문이 반드시 나온다.
그만큼 중요한 부분이라는 것.

4장. Javascript OOP

개발자가 선언하는 객체

I. Object Literal — 객체를 만들 때 간단하게 사용하는 방법

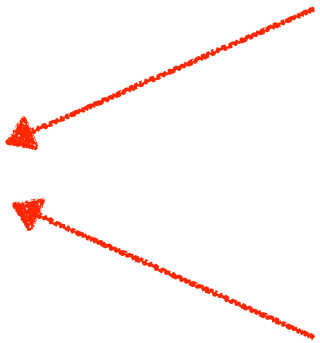
II. 생성자 함수

III. 프로토타입

IV. 다양한 OOP 기법

V. 클래스

VI. 클로저



모형을 만들어
객체를 만드는 방법

04-1. Object Literal

JavaScript Object Oriented Programming

객체 리터럴

- 객체를 만드는 방법은 여러가지가 있는데 가장 간단한 방법이 객체 리터럴(Object Literal)을 이용하는 방법입니다.

멤버라고 부르기도 한다.

- 객체 리터럴은 객체의 내용을 중괄호({ }) 로 묶고 그 안에 객체의 내용을 프로퍼티로 등록을 합니다.
- 객체의 프로퍼티는 키:값 형태를 띄며 여러 프로퍼티를 콤마(,) 로 구분해서 등록하는 방식이 객체 리터럴입니다.

```
let user = {
  name: '홍길동',
  age: 20,
  isMember: true,
  order: {
    productId: 2,
    count: 10
  }
}
```

객체 리터럴

멤버 참조 연산자

- 객체내에 선언된 것을 객체의 멤버라 표현하며 객체의 멤버 접근은 DOT(.) 을 이용합니다.
- "객체명.프로퍼티명" 형태이며 user 객체의 name 프로퍼티에 접근하는 코드는 user.name 입니다.
- 또한 user 객체의 order 프로퍼티가 객체이고 이 order 객체의 productId 에 접근하려면 user.order.productId 입니다.

```
console.log(user.name)//홍길동
console.log(user.order.productId)//2
```

- 함수도 데이터로 활용이 가능함으로 프로퍼티에 함수 대입도 가능합니다.

```
sayHello: function(){
  console.log(`Hello, ${this.name}`)//Hello, 홍길동
}
```

프로퍼티

- 객체지향 프로그래밍에서 객체에 포함된 변수와 함수를 부르는 용어가 프로퍼티, 메서드입니다.
- 용어임으로 변수와 프로퍼티를, 함수와 메서드를 혼용해서 부르고 있지만 의미가 다른 용어입니다.
- 변수는 데이터를 의미합니다.
- 변수를 하나 선언하고 그 변수에 데이터를 저장해서 이용하겠다는 의도입니다.
- 그런데 객체에 선언된 변수도 데이터를 저장하고 이용하지만 그 변수 하나에 의미가 있는 것이 아니라 그 변수의 값으로 객체의 어떤 특성을 표현하고자 하는 것입니다.
- 예를 들어 사람을 표현하기 위한 User 라는 객체가 만들어진다면 그 객체에 선언된 name 이라는 변수는 User 의 이름을 표현하기 위해 사용되는 것입니다.
- 즉 혼자서 의미가 있는 것이 아니라 자신의 값으로 객체를 표현하고자 하는 의도입니다. 그럼으로 객체의 변수를 프로퍼티라고 부르는 것입니다.

객체 리터럴

객체의 this

- 객체내에 선언된 함수에서 객체에 선언된 다른 멤버를 이용하는 경우가 있습니다.
- 이때는 this로 프로퍼티를 지정해야 합니다.
- this 는 예약어로 어떤 객체내에서 객체 자신을 지칭하는 예약어입니다.

```
let user = {
  name: '홍길동',
  age: 20,
  isMember: true,
  sayHello: function(){
    console.log(`Hello, ${this.name} - age = ${age}`)//error
  }
}
```

작동하지 않음.
객체 내에서 다른 멤버를 지칭할 때는
반드시 'this' 를 사용해야 한다.

객체 리터럴

객체의 this

- 함수는 function 예약어로 선언될 수도 있고 화살표 함수로 선언될 수도 있습니다.
- function 으로 함수를 등록하면 this 는 등록하는 객체를 지칭하지만 화살표 함수로 등록을 하면 this 는 함수가 등록된 객체를 지칭하지 못합니다.

화살표 함수에서의 this

```
1   let user = {
2     name: '홍길동',
3     age: 20,
4     sayHello1: function() {
5       console.log(`${this.name}, ${this.age}`)
6     },
7     sayHello2: () => {
8       console.log(`${this.name}, ${this.age}`)
9     }
10  }
11
12  user.sayHello1()//홍길동, 20
13  user.sayHello2()//, undefined
```

객체 리터럴

축약형으로 프로퍼티 등록

- 객체 리터럴에 값을 등록할 때 객체 외부에 선언된 변수의 값도 대입이 가능합니다.
- 프로퍼티 명과 대입되는 변수명이 동일한 경우 축약형으로 등록이 가능합니다.

함수 외부에 선언된 변수의 값을 대입

〈전역 변수〉
let name = 'kim';
let age = 20;

```
let user = {
  name,
  age,
  sayHello: function () {
    console.log(`${this.name}, ${this.age}`);
  },
};
```

name : name의 축약형

객체 리터럴

외부에서 객체 멤버 등록

- 객체 리터럴은 { } 안에 객체 멤버를 등록합니다.
- 그런데 객체를 선언하고 이후에 그 객체에 멤버를 추가할 수도 있습니다.

외부에서 멤버 등록

```
1   let user = {
2     name: '홍길동',
3     sayHello1: function() {
4       console.log(`${this.name}, ${this.age}`)
5     }
6   }
7   user.age = 20
8   user.sayHello2 = function(){
9     console.log(`${this.name}, ${this.age}`)
10  }
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare