**1. call, apply, bind 함수에 대해 설명하시오.**

자바스크립트에서 object를 효율적으로 사용하면서 중복을 방지하고 재사용 패턴까지 구현할 수 있는 유용한 방법으로 call, apply, bind가 있습니다.

call과 apply부터 살펴보겠습니다. 원래 함수는 선언한 후 호출해야 실행되죠. 호출하는 방법으로는 함수 뒤에 ()를 붙이는 것과, call 그리고 apply하는 방법이 있습니다.

함수.call(지정할 객체명, 전달할 매개변수)

함수.apply(지정할 객체명, [전달할 매개변수])

var example = function (a, b, c) {

  return a + b + c;

};

example(1, 2, 3);

example.call(null, 1, 2, 3);

example.apply(null, [1, 2, 3]);

보면 call은 보통 함수와 똑같이 인자를 넣고, apply는 인자를 하나로 묶어 배열로 만들어 넣는 것을 알 수 있습니다. 그렇다면 call과 apply가 공통적으로 가진 null 인자의 역할은 뭘까요?

바로 this를 대체하는 겁니다.

var obj = {

  string: 'zero',

  yell: function() {

    alert(this.string);

  }

};

var obj2 = {

  string: 'what?'

};

obj.yell(); *// 'zero';*

obj.yell.call(obj2); *// 'what?'*

마지막 줄에서 obj.yell.call(obj2)로 this가 가리키는 것을 obj에서 obj2로 바꾸었습니다. yell은 obj의 메소드인데도 zero 대신에 what?이 alert되었습니다. 즉 다른 객체의 함수를 자기 것마냥 사용할 수 있는 겁니다.

위 메소드들을 쓰는 예로, 함수의 arguments를 조작할 때 사용합니다. arguments는 함수라면 처음부터 갖고 있는 숨겨진 속성인데요. 바로 함수에 들어온 인자를 배열 형식으로(배열은 아닙니다. 유사 배열이라고 부릅니다.) 반환합니다.

function example2() {

  console.log(arguments.join());

}

example2(1, 'string', true); *// Uncaught TypeError: arguments.join is not a function*

에러가 발생하죠? arguments는 모양만 배열이지 실제 배열이 아니라서 배열의 메소드를 쓰면 에러가 발생합니다. 이 때 바로 call이나 apply가 효력을 발휘합니다.

function example3() {

  console.log(Array.prototype.join.call(arguments));

}

example3(1, 'string', true); *// '1,string,true'*

이제 제대로 작동하네요. 배열의 프로토타입에 있는 join 함수를 빌려 쓰는겁니다. this는 arguments를 가리키게 하고요. join 외에도 slice, concat 등등 모든 메소드를 이 방식으로 사용할 수 있습니다.

이제 bind 함수에 대해 알아보겠습니다. bind 함수는 함수가 가리키는 this만 바꾸고 호출하지는 않는 겁니다. 위의 예를 재사용하겠습니다.

var obj = {

  string: 'zero',

  yell: function() {

    alert(this.string);

  }

};

var obj2 = {

  string: 'what?'

};

var yell2 = obj.yell.bind(obj2);

yell2(); *// 'what?'*

obj.yell.bind(obj2) 했더니 yell 함수의 this가 obj2로 바뀌었습니다. 즉 call이나 apply와 비슷하지만 호출은 하지 않지 않고 함수만 반환하는 겁니다. call(this, 1, 2, 3)은 bind(this)(1, 2, 3)과 같죠.