NHN NEXT 2014-2

WEB UI Advanced Homework #2 131053 이원효

call과 apply

함수에는 apply()와 call() 두 가지 메서드가 존재한다. 이 메서드들은 모두 소유자인 함수를 호출하면서 this를 넘기는데, 결과적으로 함수 내부에서 this 객체의 값을 바꾸는 것과 마찬가지이다. apply() 메서드는 매개변수로 소유자 함수에 넘길 this와 매개변수 배열을 받는다. 두 번째 매개변수는 Array의 인스턴스일 수도 있고, arguments 객체일 수도 있다.

```
function sum(num1, num2) {
	return num1 + num2;
}

function callSum1(num1, num2) {
	return sum.apply(this, arguments); // arguments 객체를 넘김
}

function callSum2(num1, num2) {
	return sum.apply(this, [num1, num2]); // 배열을 넘김
}

alert(callSum1(10, 10));  // 20
alert(callSum2(10, 10));  // 20
```

위 예제에서 callSum1()은 sum()을 호출하면서 자신의 this와 arguments 객체를 매개변수로 넘기게 된다. callSum2() 역시 sum()을 호출하지만, arguments 객체가 아닌 매개변수의 배열을 넘기게 된다. 두 함수 모두 올바르게 실행됨을 알 수 있다.

call() 메서드도 apply()와 마찬가지로 동작하지만 매개변수를 전달하는 방식이 다르다. this가 첫 번째 매개변수인 점은 같지만, call()을 사용할 때는 반드시 다음 예제와 같이 매개변수를 각각 나열해야 한다.

```
function sum(num1, num2) {
	return num1 + num2;
}

function callSum(num1, num2) {
	return sum.call(this, num1, num2);	// this == Window
}

alert(callSum(10, 10));	// 20
```

결과는 apply()와 마찬가지이다. apply()와 call() 중 무엇을 사용할지는 순전히 개발자의 몫이며, 매개변수를 전달하기 편리한 방식을 택하면 된다. arguments 객체를 그대로 전달해도 되거나, 매개변수로 전달할 데이터가 이미 배열 형태로 준비되어 있다면 apply()가 낫고, 그렇지 않다면 call()이 더 낫다. 전달할 매개변수가 없다면 두 메서드는 완전히 동일하다.

물론 apply()와 call()의 진가는 매개변수를 전달해 호출하는 것이 아니라 this를 바꾸는 데 있다. 다음 예제를 보자.

```
window.color = “red”;
var o = { color: ”blue” };

function sayColor() {
	alert(this.color);
}
sayColor();  // red

sayColor.call(this);	// red.	this == window
sayColor.call(window);	// red.	this == window
sayColor.call(o);	// blue.	this == Object (o)
```

sayColor()는 전역 함수로 정의되어 있으므로, 전역 스포크에서 호출하면 this.color를 window.color로 평가하기 때문에 “red”를 표시하게 된다. sayColor.call(this)나 sayColor.call(window)와 같이 호출하면 함수 컨텍스트를 명시적으로 전역 스코프로 지정하는 것이므로 역시 “red”를 표시하게 된다. 하지만 sayColor.call(o)는 함수의 컨텍스트를 o로 설정하므로 “blue”를 표시한다.
call()이나 apply()를 써서 스코프를 바꾸면 객체마다 일일히 메서드를 등록하지 않아도 되는 장점이 있다. this 객체를 설명할 때는 o 객체에 직접 sayColor() 메서드를 등록해야 하지만, 이번에는 그럴 필요가 없었다.

관련하여 bind() 라는 메서드도 존재한다. 이 메서드는 새 함수 인스턴스를 만드는데, this는 bind()에 전달된 값이 된다. 다음의 코드를 보자.

```
window.color = “red”;
var o = { color: “blue” };

function sayColor() {
	alert(this.color);
}

var objectSayColor = sayColor.bind(o);
objectSayColor();   // blue.	this == Object (o)
```

이 예제에서는 sayColor()에서 bind()를 호출하면서 객체 o를 넘겨 objectSayColor()라는 함수를 생성한다. objectSayColor() 함수의 this는 o에 묶이므로 전역에서 함수를 호출하더라도 항상 “blue”를 표시하게 된다.
(bind()를 지원하는 브라우저: IE 9+, Firefox 4+, Safari 5.1+, Chrome)

함수로서의 생성자에서의 call()과 apply()

생성자 함수와 다른 함수의 차이는 호출 방식이다. 생성자는 결국 함수이고, 함수가 자동으로 생성자처럼 동작하게 만드는 특별한 문법은 없다. new 연산자와 함께 호출한 함수는 생성자처럼 동작하게 된다.
new 연산자 없이 호출한 함수는 일반적인 함수에서 예상하는 것과 똑같이 동작한다.

```
// 생성자로 사용
var person = new Person(“Woonohyo”, 27, “Software Engineer”);
person.sayName();                    // “Woonohyo”

// 함수로 호출
Person(“Nigayo”, 3x, “Professor”);   // window에 추가됨
window.sayName();                    // “Nigayo”

// 다른 객체의 스코프에서 호출
var o = new Object();
Person.call(o, “Aragaki Yui”, 27, “Actress”);        // this == o
o.sayName();                                         // “Aragaki Yui”
```

예제의 첫 번째 부분은 일반적인 생성자 패턴으로, new 연산자와 함께 사용해서 새로운 객체를 생성한다. 두 번째 부분은 Person() 함수를 new 연산자 없이 호출한 경우이다. 프로퍼티와 메서드는 window 객체에 추가된다. 함수를 호출할 때 객체의 메서드로서 호출하거나 call() / apply() 를 통해 호출해서 this의 값을 명시적으로 지정하지 않을 경우, this 객체는 항상 Global 객체(웹 브라우저에서는 window)에 묶이게 된다. 따라서 함수를 호출하면 sayName() 메서드를 window 객체에서 호출할 수 있고, “Nigayo”가 반환된다.
call()이나 apply()를 통해 Person() 함수를 다른 객체의 스코프에서 호출할 수 있다. 이 경우 함수의 this 값은 객체 o가 되며, 객체 o는 프로퍼티 전부와 sayName() 메서드를 할당받는다.