공학 트렌드

자바스크립트(Javascript) UI Part 2
- 재사용 패턴

2012. 12. 17. (제37호)

목 차

소프트웨어공학센터
경영지원TF팀

Ⅰ. 코드 재사용

코드 재사용은 대부분의 언어에서 필요한 기능이다. 비슷한 요구사항에 맞춰서 동일한 기능을 반복적으로 개발하는 것은 중복코드를 증가시켜 결국 낭비로 이어진다. UI개발과정에서도 이런 문제는 존재하며 다른 언어와 마찬가지로 그 해법도 크게 다르지 않다.

UI개발의 경우는 재사용을 위해서 자바스크립트 코드를 재사용할 수 있고, CSS Framework(less나Sass등)을 통한 CSS 코드를 재사용하는 방법들도 존재한다.

자바스크립트의 경우 객체 지향적으로 모듈화를 통해 개발 할 수 있음에도 불구하고, Java와 같이 'extend'와 같은 명시적인 상속을 위한 기능이 존재하지 않는다. 다행히 이런 부분이 최근 ECMAScript 6[1]에 반영되고 있는 중이다.

물론 지금 상속을 구현하기 위해서 아무것도 할 수 없는 것은 아니다. 이미 대부분의 자바스크립트 라이브러리와 이를 기반으로 제작된 많은 UI 콤포넌트들은 'prototype' 이라는 속성을 통해 상속을 구현하고 있으며, 결과적으로 이것을 활용해 코드 재사용이라는 목적을 달성할 수 있다.

본 문서에서는 자바스크립트의 대표적인 재사용패턴을 설명하며, 그에 앞서 prototype[2]이라는 자바스크립트 코어에 해당하는 핵심 개념을 설명한다.

Ⅱ. 왜 Prototype 인가?

다음의 코드를 살펴보자.

```
function Parent() {
   this.getName = function(){}
   this.fixName = "hary"
}

var oP1 = new Parent();
var oP2 = new Parent();
var oP3 = new Parent();
```

코드 1

1) http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts
2) 이 문서에의 prototype은 *prototype* 이라는 자바스크립트 라이브러리와는 다른 개념이다.

Parent라는 함수를 new 키워드를 통해서 호출했다. 이와 같이 생성된 함수를 생성자(constructor)라고 하며, 그때 생성된 oP1,oP2,oP3과 같은 인스턴스는 'this'라는 자바스크립트 키워드를 통해 지정된 속성들을 사용할 수가 있게 된다.

이렇게 인스턴스를 여러 개 생성하여 사용하는 경우 아래의 문제가 발생한다.

인스턴스마다 가지고 있는 함수가 모두 다른 것들이 된다. 즉 인스턴스 별로 각각 개별적인 함수를 각각 가지게 되는 것이다.

아래 비교문을 통해 이를 확인해보자.

oP1.getName === oP2.getName //false
oP2.getName === oP3.getName //false

이런 경우 실제로 메모리에 서로 다른 함수가 계속 증가하게 된다. 또한 인스턴스마다 프로퍼티가 늘어나는 비용도 함께 발생한다. 자세한 확인은 브라우저 디버깅 도구를 통해서 oP1이나 oP2와 같은 인스턴스 값을 확인해보면 자세히 알 수 있다.

특히 큰 규모의 자바스크립트 라이브러리들이 이와 같은 구조로 되어 있다면 개발자(라이브러리 사용자)입장에서는 좀 더 비용문제가 심각해 질 수도 있다.

prototype이라는 것은 아래와 같이 활용할 수가 있으며, 위의 코드를 개선해서 비용이 절감된 재사용 패턴을 구현 할 수가 있다.

```
function Parent() {}
Parent.prototype.getName = function() {}
Parent.prototype.fixName = "hary"

var oP1 = new Parent();
var oP2 = new Parent();
var oP3 = new Parent();

oP1.getName === oP2.getName //true
oP2.getName === oP3.getName //true
```

코드 2

위 코드에서는 Parent함수에 prototype키워드를 활용해서 각 메서드를 속성으로 추가했다. 그리고 그 결과 인스턴스별 메서드는 모두 같은 참조를 가지고 있다. 일종의 메모를 공유하고 있는 셈이다. 이것은 코드1과 다른 결과이다.

이것이 prototype 객체가 가진 속성 중 가장 대표적인 특징이기도하다. 이런

특징을 활용해서 객체간의 상속을 통해 재사용 효과를 얻을 것이다.(잊지 말아야 할 것은 재사용패턴은 코드의 재사용을 줄이는 것이고, 결국 비용을 줄일 수 있어야 한다.)

코드2를 다시 살펴보면 prototype은 Parent함수의 속성(프로퍼티)이다. 반대로 말하면 prototype은 Parent에 속해있는 객체(object)이다. 실제로 모든 함수는 함수가 생성됨과 동시에 prototype이라는 객체를 가지고 있다.

prototype을 사용하고 접근하기 위해서는 prototype에 메서드와 같은 속성을 추가해줘야 하며, (코드 2에서 fixName과 getName()) 이를 쉽게 활용하기 위해서는 'new' 연산자를 활용해야 한다. 자바스크립트의 'new' 연산자를 통해 함수를 호출하면 이미 존재하는 prototype객체에 직접 접근 할 수가 있게 된다. 이 과정에서 내부적으로 [[Prototype]] 이라는 숨겨진 객체를 통해서 접근하게 된다. (크롬 브라우저의 디버깅 도구에서는 '__proto__' 라는 속성으로 확인 할 수 있다. 이것은 아직 비표준이며 표준화를 위해 논의 중인 단계이다)

그림 1에서는 Parent함수를 통해서 생성된 oP인스턴스를 통해서 getName 메서드에 접근하는 것을 도식화 했다.

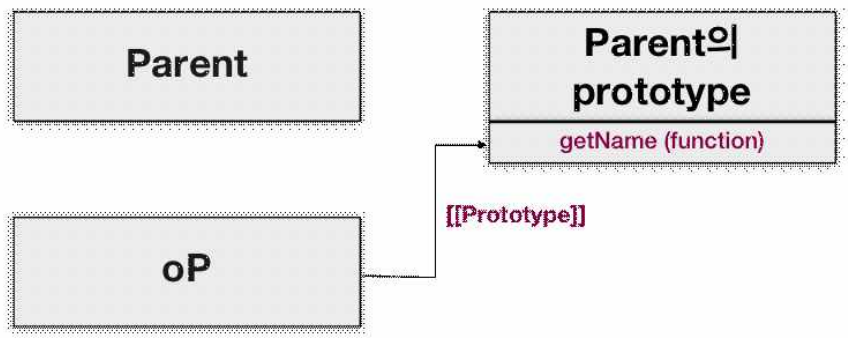

그림 1 인스턴스를 통한 프로토타입 객체 접근

Ⅲ. prototype을 통한 상속

prototype 객체의 특성 중 한 가지는 모든 'prototype객체는 다른 prototype객체를 가지고 있다'는 것이다. 기본적인 prototype객체는 Object.prototype를 내부에서 가지고 있다. 이것은 아래와 같은 구조로 설명 할 수 있다.

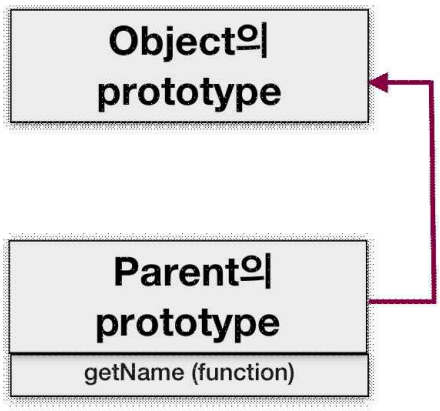

그림 2 Prototype chain

이와 같이 prototype 간의 관계를 prototype chain이 형성되었다고 한다.

prototype chain은 객체의 속성을 찾는 순서로 설명할 수 있는데, parent의 prototype에 찾는 메서드나 함수가 없다면, 그 다음으로 Object의 prototype에서 속성을 찾게 된다.

prototype chain 은 사용자에 의해서 추가 할 수도 있는데, 자바스크립트 상속은 바로 이 prototype chain을 활용해서 구현 할 수가 있다.

이제 Child함수를 만들고 역시 prototype을 선언해보자. 이때 Child의 prototype을 부모의 인스턴스를 가리키도록 한다. 부모의 인스턴스를 가리킨다는 의미는 부모의 prototype에 접근 할 수 있다는 이야기다.(앞선 코드에서 parent의 prototype의 접근은 parent의 인스턴스인 oP를 생성함으로써 접근 했었다)

코드를 살펴보면 아래와 같다.

```
function Parent() { }
Parent.prototype.getName = function() {return 123 };

//Child 함수와 prototype을 생성
function Child() { }
Child.prototype = new Parent();

var oC = new Child();
console.log(oC.getName()); // 456
```

코드 3

코드3 에서 Child의 prototype은 실제 Parent의 인스턴스 역할을 하고 있는 셈이다. 다시 말해 'Child.prototype == oP' 와 같은 의미이며, 결국 oC를 통해서 Parent의 prototype객체까지 접근 할 수가 있게 된다.

코드3은 그림3과 같이 parent와 child 와의 관계를 표현 할 수 있다.

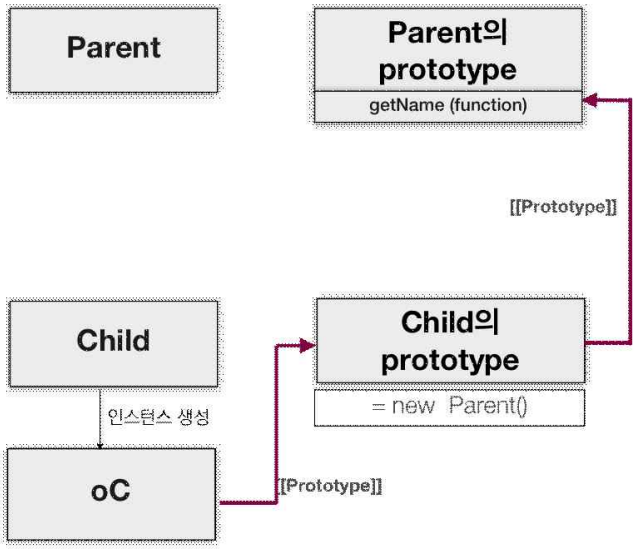

그림 3 prototype inheritance

한 가지 주의 할 점은 Child.prototype 에 부모의 인스턴스가 아닌 Parent의 Prototype을 직접 할당하면 복잡한 문제가 발생한다. 부모의 프로퍼티가 원치 않게 쉽게 변경이 될 수 있음으로 이렇게 사용하지 않는 것이 좋다. 부모의 Prototype을 직접 받아서 사용하는 개선된 방법은 다음 절에서 확인 할 수가 있다.

IV. Object.create()를 활용한 Prototypal inheritance

Prototypal inheritance라고 하는 방식은, 기존 방법과 달리 상속을 위해 부모의 인스턴스 생성을 하지 않아도 된다. 이런 부모의 인스턴스 생성과정에서의 추가적인 문제는, 부모함수에서 인자의 개수를 체크하는 로직이 포함되어 있다면 인자 없이 부모를 호출하는 과정에서 문제를 일으키게 된다.

ECMAScript5 에서는 Object에 create라는 메서드를 포함하고 있다. (ECMAScript5는 IE 9미만의 브라우저를 지원하지 않는다)[3)]

Object.create()는 prototype 객체를 통한 상속을 위한 아주 간단한 메서드이다.

3) http://kangax.github.com/es5-compat-table/

```
function create(o) {
   function F() {}
   F.prototype = o;
   return new F();
}
```

코드 4 Object.create 내부 소스코드(브라우저 마다 조금씩 다를 수 있음)

코드4에서 create 메서드는 새로운 인스턴스를 반환하고 있다. 그리고 인자로 받는 값은 부모의 prototype객체이다. 이 부분은 코드3에서의 부모의 인스턴스를 생성한 것과 달리, 부모의 prototype을 받음에도 불구하고 부모의 메서드가 함부로 바뀌는 것을 걱정하지 않아도 되는 코드이다.

이것을 가능하도록 create 메서드에서는 부모의 prototype을 전달받아 이를 중간의 프록시 개념의 역할을 하는 임의의 함수(코드4의F())를 이용해서 전달해주고 있기 때문에 가능하다.

Object.create()를 활용해서 구현 한 예제코드는 아래와 같다.

```
function Parent() {}
Parent.prototype.getName(){}

function Child() {}
Child.prototype = Object.create(Parent.prototype);

var oC = new Child();
oC.getName();
```

코드 5

그림4에서는 코드5의 관계를 표현했다.

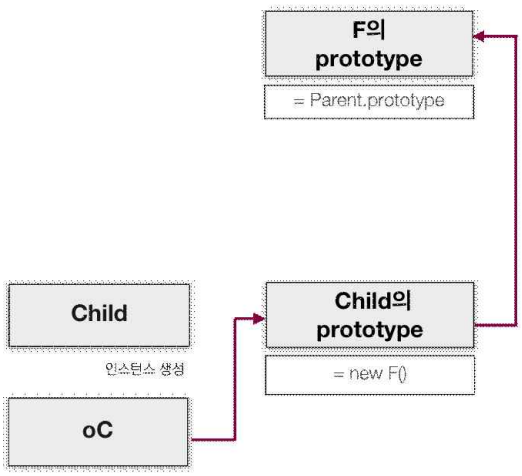

그림 4 Object.create()를 통한 상속

Object.create()에서 사용한 방법은 prototype을 가장 잘 활용한 방법으로 알려져 있으며(Prototypal inheritance)실제로 가장 널리 활용되는 방법이기도하다. Object.create()를 지원하지 않는 브라우저에서도 간단히 Object.create()를 구현함으로써 하위호환성을 유지할 수가 있다.

Ⅳ. 결론

UI개발에 필요한 상속의 개념을 이해하는 것은 다소 복잡한 편이다. 그리고 간단한 코드를 작성하는데 prototype을 활용하여 상속을 적용하는 것도 적절한 선택은 아니다. 하지만 많은 컴포넌트를 사용하거나, 표현해야 할 웹 UI 객체가 많은 상태라면, 상속을 통한 효율적인 재사용패턴을 활용할 수 있다.

또한 대부분의 현대의 자바스크립트 라이브러리에서는 이러한 상속방법을 지원하며, Prototypal inheritance 방식과 유사한 방식을 추상화하여 구현해두고 있다. 이를 잘 활용해서 재사용을 통한 비용감소 효과를 얻는 것에 관심을 가지는 것도 좋다.

더불어 자바스크립트의 prototype객체라는 코어개념을 이해하게 되면, 상속과 관련된 문제뿐 아니라, 다양한 UI 개발에서의 문제를 쉽게 해결 할 수가 있을 것이다.

<참고 자료>

1. JavaScript Patterns. Stoyan stefanov, 2010
2. JavaScript: The Good Parts. Douglas Crockford. 2008
3. Pro JavaScript Techniques. John Resig. 2006
4. http://www.jspatterns.com
5. http://javascript.crockford.com/prototypal.html
6. http://www.2ality.com/2012/01/js-inheritance-by-example.html