[생성자 함수를 통한 상속]

Prototype과 \_\_proto\_\_ 에 대해서 알고 있다는 가정하에 설명한다.

물론 좀 모르고, 개념이 완벽히 정립되지 않았더라도 적당히 알 수 있을 것이다.

ES6까지 온 지금은 class를 이용해서 상속 할 수 있다. 물론 Original javascript constructor function을 이용해서 상속할 수도 있고 말이다.

이번 시간에는 original 한 방법을 통해서 상속을 해보도록 한다.

근데 실제로는 class를 많이 쓴다고도 한다. 일단 original한 방법을 알아보도록 하자.

class Person{

    constructor(name, first, second){

        this.name = name;

        this.first = first;

        this.second = second;

    }

    sum(){

        return (this.first + this.second);

    }

}

class Personplus extends Person{

    constructor(name, first, second, third){

        super(name, first, second);

        this.third = third;

    }

    sum(){

        return super.sum()+this.third;

    }

    avg(){

        return (this.first+this.second+this.third)/3;

    }

}

자 class로써 상속을 구현하는 방법이다. 다음 페이지에 original 방법을 보인다.

function Person(name, first, second){

    this.name = name;

    this.first = first;

    this.second = second;

}

Person.prototype.sum = function(){

    return this.first + this.second;

}

function PersonPlus(name, first, second, third){

    Person.call(this, name, first, second);

    this.third = third;

}

PersonPlus.prototype = new Person();

PersonPlus.prototype.avg = function(){

    return (this.first+this.second+this.third)/3;

}

이렇게 한다.

내용이 복잡할 수 있어서 일단 전체를 보이고 코드설명을 하는게 낫겠다.

function Person(name, first, second){

    this.name = name;

    this.first = first;

    this.second = second;

}

첫번째 부분이다. 이 부분은 크게 어렵지 않을 것이라고 생각한다. 부모 constructor function을 만드는 것이다.

Person.prototype.sum = function(){

    return this.first + this.second;

}

이 부분 역시 크게 어렵지 않을 것이다. 그냥 함수를 추가하는 것이니까. 왜 Person.prototype에 sum을 추가하는 것인지는 이전 글들을 봐주면 좋겠다.

function PersonPlus(name, first, second, third){

    Person.call(this, name, first, second);

    this.third = third;

}

자 대망의 자식 constructor function인 PersonPlus 이다. 이 PersonPlus는 name, first, second로 파라미터가 끝나는 부모 function인 Person과 다르게 third 까지 받는다.

function PersonPlus(name, first, second, third){

    this.first 는 어쩌구..

    this.second = second...

    this.third = third;

}

단순 값을 저장하는 변수 프로퍼티들을 지정하는 부분은 부모와 함께 공통적으로 이렇게 들어가도 괜찮다. 하지만 중복이므로, 제거하는 것이 좋다.

Person(name, first, second);

부모 function인 Person은 함수이므로, this.name, first, second를 다 설정해주는 방법으로 이렇게 호출하면 어떨까?

아쉽지만 이렇게 해선 구현할 수 없다. 왜냐하면 다음페이지에..

function Person(name, first, second){

    this.name = name;

    this.first = first;

    this.second = second;

위와 같이 this.name = name이기 때문이다. 무슨 말이냐면 PersonPlus 객체에서 Person을 호출한다 쳐도 Person에서 실행되는 this.name의 this라는 부분이 PersonPlus를 의미하진 않기 때문이다.

따라서 우리가 저번에 배웠었던 call() 함수를 쓴다. Call 함수는 모든 함수 객체에 들어있으며, 해당 함수 내에서 실행되는 this 지정어를 call() 함수의 매개변수로 하게해주는 함수이다.

function PersonPlus(name, first, second, third){

    Person.call(this, name, first, second);

    this.third = third;

}

이렇게 Person.call로 Person을 호출할 때 this를 PersonPlus로 하여 호출했다. 이렇게 설정되는 것이다.

그리고, 상속을 들어간다.

상속이란건 prototype 관점에서 무엇을 의미하는가?

왼쪽 그림을 보자.

그림을 좀 못그리긴 했지만, 결국 상속이란 자식 프로토타입 객체가 \_\_proto\_\_로 부모 프로토타입 객체를 참조하는 것이다.

그 말은 무엇인가? 저렇게 구현하려면 어떻게 해야하는가?

A라는 부모 constructor function의 프로토타입을 \_\_proto\_\_로 참조하는 객체.. 그것은?

A로 만들어진 객체가 그러하다!!!!!

PersonPlus.prototype = new Person();

따라서 위와 같이 만들어줄 수 있다.

그런데 이렇게 하면, Person(매개변수)에서 매개변수 값을 못줬으므로,

Person {name: undefined, first: undefined, second: undefined, avg: ƒ}

PersonPlus.prototype을 확인했을 때(Person 객체이다) undefined라며 정의되지 않은 값들을 객체에 저장하고 있게 된다. 이는 그리고 데이터 낭비다. 물론 얼마 낭비하는게 아니긴 하지만.

아무튼 깔끔한게 좋으니까 아래와 같이 하도록 한다.

PersonPlus.prototype = Object.create(Person.prototype);

console.log(PersonPlus.prototype);

//Person {}

이렇게 말이다. log떠서 확인해보면? 텅 빈 Person 객체라는 것을 나타내준다. 아 참고로 \_\_proto\_\_로는 당연히 Person.prototype을 가리킨다.

PersonPlus.prototype.avg = function(){

    return (this.first+this.second+this.third)/3;

}

그렇게 상속을 구현하고, avg를 추가한다. 이렇게 자식 프로토타입에 새로운 메소드를 추가할 때에는 상속을 먼저하는게 옳다. 그래야 prototype 덮어쓰기가 안되니까. Avg를 만들고 상속을 하면 Object.create(부모)로 만들어진 부모 객체가 avg를 가지고있는 자식 프로토타입을 덮어쓰게 되지 않은가?

그 다음 새로운 sum을 정의해보자. 부모 constructor function인 Person의 sum은 first와 second만 더해주는 것이니까, third가 있는 PersonPlus에서는 third 까지 더해주는 함수를 구현하고 싶다.

PersonPlus.prototype.sum = function(){

    //return this.first + this.second + this.third;

    //return this.\_\_proto\_\_.\_\_proto\_\_.sum.call(this) + this.third;

    return PersonPlus.prototype.\_\_proto\_\_.sum.call(this) + this.third;

}

당연히 이렇게 그냥 평범하게 프로토타입에 함수 넣어주면 된다.

참고로 주석문은 다 똑같은 동작을 한다는 것을 의미한다.

여기서 this는 PersonPlus로 만들어진 객체를 의미할 것이다. 그리고 부모 prototype(Person.prototype)의 sum을 불러와 그 값에 this.third를 더해야 하므로 Person.prototype.sum을 콜하는데, 이걸 부르는 방법을 달리한 것일 뿐이다.

return Person.prototype.sum.call(this) + this.third;

당연히 이렇게 해도 된다. 근데 이건 좀 너무 하드코딩 같은 느낌?

var kim = new PersonPlus('kim', 10, 20, 30);

아무튼 그렇게 해놨으면 이렇게 kim 객체를 PersonPlus 객체로 하여 만들면 된다.

var kim = new PersonPlus('kim', 10, 20, 30);

console.log("kim.sum()", kim.sum());

//kim.sum() 60

console.log("kim.avg()", kim.avg());

//kim.avg() 20

상속도 잘되어있다. 아래에 최종 코드를 적는다.

function Person(name, first, second){

    this.name = name;

    this.first = first;

    this.second = second;

}

Person.prototype.sum = function(){

    return this.first + this.second;

}

function PersonPlus(name, first, second, third){

    Person.call(this, name, first, second);

    this.third = third;

}

PersonPlus.prototype = Object.create(Person.prototype);

PersonPlus.prototype.avg = function(){

    return (this.first+this.second+this.third)/3;

}

PersonPlus.prototype.sum = function(){

    //return this.first + this.second + this.third;

    //return this.\_\_proto\_\_.\_\_proto\_\_.sum.call(this) + this.third;

    //return Person.prototype.sum.call(this) + this.third;

    return PersonPlus.prototype.\_\_proto\_\_.sum.call(this) + this.third;

}

var kim = new PersonPlus('kim', 10, 20, 30);

console.log("kim.sum()", kim.sum());

//kim.sum() 60

console.log("kim.avg()", kim.avg());

//kim.avg() 20

그렇게 코드가 길지도 않았네..

근데 아직 하나 더 남았다 안전띠 꽉 매라. Constructor 관련이다.

[constructor 속성]

이번엔 constructor 프로퍼티에 대해서 생각해보자.

function aaa(){

}

console.log(aaa.prototype);

우리가 이렇게 출력하면 aaa 함수의 prototype이 출력된다.

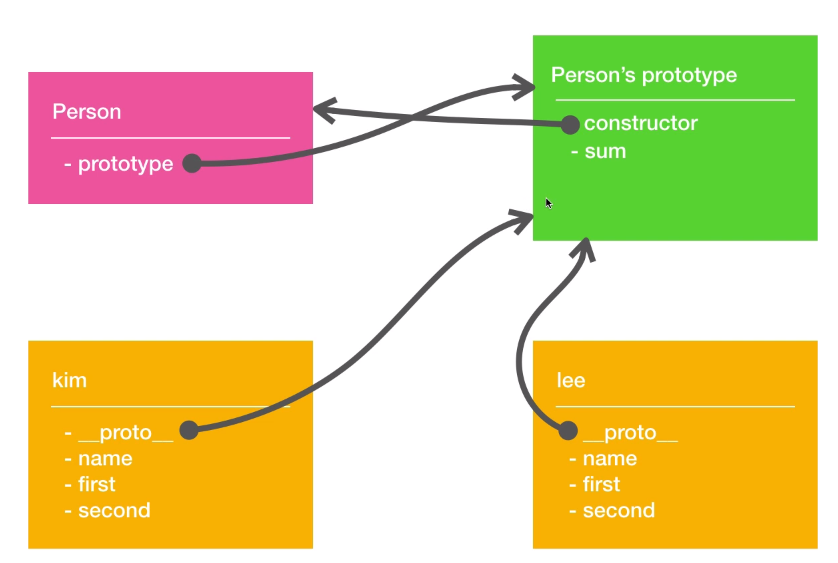
텍스트이(가) 표시된 사진

자동 생성된 설명개발자 도구로 보면 이런 모양이다. 그런데 constructor라는 프로퍼티가 보이는가? 저게 바로 이번에 얘기해볼 constructor 프로퍼티다.

저 프로퍼티는 f aaa() 즉, function aaa()를 가리키고 있다. aaa.prototype의 constructor, 즉 생성자 function은 aaa라는 것이다. 어찌보면 자명하다. Aaa.prototype은 object라는 function이 만들긴 했지만 aaa가 생성되면서 생성된 것이다. Object는 머리에서 지우고, 아무튼 그렇게 서로가 서로를 지정하는 관계라는 것이다.

Aaa.prototype으로 prototype 객체를 호출할 수 있고, aaa.prototype.constructor로 aaa()를 호출할 수 있다.

즉, 상호참조하는 상태다.



이고잉이 준 그림이다. 이런 식이다.

그런데 여기서 kim.constructor 하면 어떤게 호출될까?

간단히 생각해보자. Kim 객체에 보면 constructor 프로퍼티는 없다. 따라서 \_\_proto\_\_를 참조한다.

Kim.\_\_proto\_\_는 Person.prototype이다. 즉 kim.constuctor라는 것은 kim.\_\_proto\_\_.constructor를 호출하는 것이다.

그것은? Kim.prototype.constructor 이므로, 그것은 Person() 이라는 함수 자체를 의미하게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명이런 예시도 있다.

D는 date 객체이다. Date 객체의 prototype의 constructor는 ? 당연히 date이다.

d.constructor는? d에게는 바로 constructor라는 프로퍼티가 없으므로 \_\_proto\_\_를 탐색한다.

따라서 d.\_\_proto\_\_.constructor를 호출하고, 그것은 Date 이다.

이렇게 constructor는 어느 객체의 출신 성분을 나타내는 효과를 준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이런식으로 쓸 수도 있다. D.constructor 는 Date 객체니까 어떤 객체와 똑같은 형식의 객체를 만들고자 한다면 해당 객체의 생성자 명을 모르더라도 constructor를 써서 만들어버리면 된다.

js에서 constructor는 여러 의미로 쓰이는데, 그 중에 하나는 바로 살펴봤듯이

어떤 함수에서 이 객체는 생성됐는가? 를 뜻한다.

동시에 새로운 객체를 constructor function이 몰라도 constructor 프로퍼티를 호출하는 것만으로 만들 수 있는 것이다.

이걸 왜 알려주나? 다음에.

[객체 상속시 constructor도 만져줘라.]

function Person(name, first, second){

    this.name = name;

    this.first = first;

    this.second = second;

}

~~어쩌구저쩌구

PersonPlus.prototype = Object.create(Person.prototype);

~~어쩌구저쩌구

var kim = new PersonPlus('kim', 10, 20, 30);

우리는 앞에서 상속을 이렇게 했다.

근데 이래버리면? Kim의 constructor는 Person이 된다.

왜냐면 kim.\_\_proto\_\_ 는 PersonPlus.prototype을 가리키고 있는데, PersonPlus.prototype은 Person 타입 객체이기때문에 constructor 프로퍼티를 가지고 있지 않으므로, PersonPlus.prototype.\_\_proto\_\_를 참조하게 된다.

근데 그것은 Person.prototype 이므로, Person.prototype.consturctor를 kim.constructor의 반환값으로 하게 된다.

근데 그건? Person 함수 그 자체이므로, kim은 PersonPlus 타입 객체임에도 Person 타입이라고 뜨는 것이다.

Kim.constructor를 PersonPlus로 표기하려면 어떻게 해야 할까.

당연히 PersonPlus.prototype의 constructor 프로퍼티가 없기 때문에 이런 문제가 발생하는 것이니까

PersonPlus.prototype.constructor = PersonPlus;

이렇게 하면 되겠다. 그냥 땜빵이다. 어려워할 것 없다. ㅋㅋㅋ.

근데 이 모든 귀찮음을 해결하는 방법이 있다.

PersonPlus.prototype = Object.create(Person.prototype);

PersonPlus.prototype.constructor = PersonPlus;

PersonPlus.prototype.avg = function(){

    return (this.first+this.second+this.third)/3;

}

위를 보라. Avg가 왜 Person.prototype = object.create() 구문 아래에 왔는 줄 아는가?

Object.create가 새로운 객체를 만들어서 PersonPlus.prototype을 덮어쓰기 때문이다.

그러니까 avg를 먼저 선언해버리면 나중에 상속할 때 덮어써지니까 avg가 사라져버린다. ㅠㅠ 그래서 avg를 나중에 선언해주는 것이다.

아무튼 이런 순서 문제, 코드의 줄(객체 만들기 + constructor 설정까지 총 2줄) 문제를 한방에 해결하는 방법이

PersonPlus.prototype.\_\_proto\_\_ = Person.prototype;

위와 같이 하는 것이긴 한데 던더 프로토(\_\_proto\_\_를 던더 프로토라고 부른다, 던더 프로토를 직접 쓰는 것은 비표준이다.)를 쓰는 이건 쓰지 않는다.

왜냐하면.. 비표준이기 때문이다 ㅠㅠㅠㅠㅠㅠㅠㅠ

근데 \_\_proto\_\_는 대부분의 브라우저에서 구현이 되어있기 때문에 알아서 선택해서 쓰면 된다.

이고잉.. 어쩌라는거냐..

아무튼 결론이 뭐냐? Class 써라. 이거였다 ㅋㅋ

그치만, javascript를 정복하고 싶다면 \_\_proto\_\_, prototype에 대해서 아는 것이 중요하다.

근데 가성비는 안좋다… 알면 뭐하냐 class 쓸텐데.. ㅋㅋ

다 읽느라 고생했다.