**JavaScript**

**[Closure]**

var arr = []

for(var i = 0; i < 5; i++){ <- 1

arr[i] = function(id) { <- 3

return function(){

return id;

}

}(i); <- 2 (인자 값이 들어갈 수 있음)

}

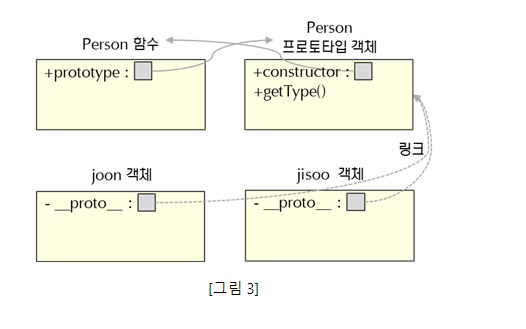
**[Prototype]**

JavaScript에서는 클래스가 없지만 Prototype을 이용하면 객체와 비슷하게 사용할 수 있다. new + 생성자 함수를 통해 만든 객체에는 모두 proto를 통해 prototype과 연결 될 수 있다. 하지만, 이는 각자 객체에서 멤버를 가지고 있는 것은 아니기 때문에 현.prototype=부모.prototype을 통해 멤버를 복사할 필요가 있다.

1. 네이티브 생성자 함수도 prototype을 통해 prototype 체인을 이용한다.
2. prototype을 통해 상속받을 수 있고 + 각자의 인자를 사용하려면 현.prototype=부모.prototype

자식은 부모 멤버를 사용할 수 있고 자식.\_\_proto\_\_는 부모.prototype과 같다. 즉, 부모의 prototype객체를 가리키게 된다.

중요한 점. 가장 가까운데 속성을 반환한다.



function Person(){}

var joon= new Person();

var jisoo= new Person();

Person.prototype.getType=function(){

return “사람”};

이렇게 되면 joon안에 person객체가 생기고 person객체안에 prototype이 protype객체와 멤버 변수들을 가리킬 수 있으므로 Person객체 안에 getType 함수가 생기는 거와 같은 효과가 생긴다.

joon.getType이 가능해진다.

근데 joon.getType=function(){

return “인간”} 으로 수정하면 joon의 getType이 추가되는 것이지 prototype 멤버 getType이 바뀌는게 아니다.

name || “재철” name이 null일 경우 “재철”이 들어간다.

return if(name==”재철”) 0 : 1;

closure 내부 함수 생성

1번의 i값이 2번의 i로 들어가고 3번의 id에 인자로 들어가게 된다.

arr[i] 가 함수 객체를 담고 있다.

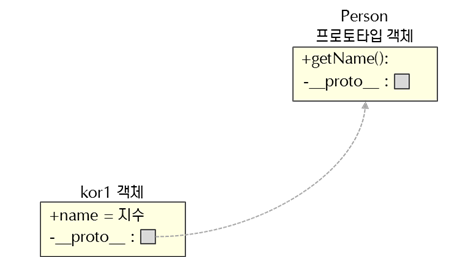
**[Var Let Const]**

var vs let vs const

var 은 재선언이 가능

let 은 재선언이 안된다. 재할당은 가능

const는 모두 불가능



function Person(name){

this.name=name

}

Person.prototype.getName=function(){ return this.name};

};

function Korean(name){

this.name=name;

}

Korea.prototype=Person.prototype;

var kor=new Korean(“재철”)

JAVA > JSTL > HTML > Javascript 순서로 동작한다.