Node -> node.js교과서 ~p85

노드는 자바스크립트의 실행기

ES2015+(ES6이라고도함)

2015년 자바스크립트 문법의 변화 이후 매년마다 문법변경사항이 발표되어 왔다 해당 변경점을

<https://node.green/> 사이트를 통해서 확인할수있다

-const와 let이 공통적으로 갖는 블록 스코프

선언된 블록안에서만 접근가능하기때문에 선언된 블록 밖에서 접근시 오류발생

const는 선언시 값을 변경하지못한다 ==>자바스크립트를 사용할때 한번 초괴화했던 변수에 다른값을 할당하는 경우는 의외로 적기때문에 변수 선언시 const를 사용한다

let은 선언후 값을 변경할수있다

-탬플릿 문자열 : ES2015문법에 생긴 새로운 문자열

문자열을 큰따옴표나 작은따옴표로 감싸는 기존의 문자열과 달리

ES2015문자열은 백틱 ` 으로 감싼다`(tab위에있는) 안의내용에 $(변수)형식으로 기호없이 문자열을 넣을수있으며 백틱을 사용하므로 큰따옴표나 작은따옴표노 문자열에 함께사용할수있다

const num1 = 1;

const num2 = 2;

const result = num1+num2;

const string = `${num1} 더하기 ${num2}는 '${result}'입니다`;

console.log(string);

결과 :

PS E:\오전수업\node\ch2> node string

1 더하기 2는 '3'입니다

PS E:\오전수업\node\ch2>

-객체 리터럴

var sayNode= function(){

    console.log('Node');

}

var es = 'ES';

var oldObject = {

    sayJS : function(){

        console.log('JS');

    },

    sayNode : sayNode,// 속성명 : 변수명

};

oldObject[es+6] = 'Fantastic'; //객체 릴터럴 바깥에서 [es+6]을 생성

oldObject.sayNode();

oldObject.sayJS();

console.log(oldObject.ES6);

const newObject = {

    sayJS(){

        console.log('JS');

    },sayNode, // 속성명과 변수명이 동일한 경우에는 한번만써도 되게 바뀌었음

    [es+6]: 'Fantastic' //객체의 속성명을 동적으로 객체 리터럴안에 생성할수있다

}

newObject.sayNode();

newObject.sayJS();

console.log(newObject.ES6);

-화살표함수

기존의 function (){}를대채= return문을 줄일수있다

const 함수명= (매개변수명,매개변수명)=>리턴결과

-구조분해할당 : 객체와 배열로부터 속성이나 요소를 쉽게 꺼낼수있다

객체명 = { 속성명1,속성명2}

const{변수명2(==속성명2),변수명1(==속성명1) }=객체명

객체안의 속성을 변수와 매칭하는데 변수명이 객체안의속성명과 이름이 같다면 해당 속성의 값을 변수명에 저장해준다

배열에대한 구조분해 할당 문법

const array = [‘nodejs’, {}, 10, true];

const[node,obj, ,bool] = array; 로 선언하게되면 변수명 node에는 ‘nodejs’가 변수명 obj에는 객체{}가 변수명 bool에는 true가 저장되게된다\

클래스문법 : 프로토타입 기반 문법을 보기좋게 클래스로 바꾼것 p73

var Human = function(type){//Human의 생성자함수를 정의하고있다

    this.type = type ||'human';//매개변수로받아온값이 있다면 Human객체의type(this.type)에 type을 저장하고 매개변수없이 생성하였다면

                                //디폴트로 human을 넣어 저장한다

};

Human.isHuman = function(human){ //Human함수 객체에새로운 메서드 isHuman을 추가하는것

    return human instanceof Human;//instatceof연사자를통해 human객체가 Human  생성자함수를 통해 생성된 객체인지 검사

}

Human.prototype.breathe = function(){ /\* 자바스크립트는 클래스 기반 언어가 아니기때문에 'prototype'을

사용하여 객체지향적인 프로그래밍을 구현 'prototype'은 생성자함수를 통해 정의된 객체의 부모객체 역활을한다 즉

'prototype'객체에 정의된 프로퍼티와 메소드는 생성자함수를 통해 생성된 모든 객체에서 공유할수있다 \*/

    console.log('haaaaaaaaaaam');

}

var Zero = function(type, firstName, lastName){

    Human.apply(this, arguments);//apply() 메서드를 사용하면 함수 내부에서 this 키워드의 값을 원하는 객체로 지정하여

    //여기서 this는 Zero객체에 Human객체의 arguments 생성자함수부분에 정의된 변수와 메서드를 가져와 전달한다 따라서

    //Zero 함수의 인스턴스로 설정되어 생성된 객체에 (Human 함수에서 정의된 프로퍼티와 메서드)arguments를 상속할 수 있다

    this.firstName = firstName;

    this.lastName = lastName;

};

Zero.prototype = Object.create(Human.prototype);

//Object.create() 함수를 사용하여 Human.prototype을 프로토타입 객체로 갖는

//새로운 객체를 생성합니다. 이 새로운 객체는 Human.prototype 객체를 상속받으므로

//, Human.prototype에 정의된 프로퍼티와 메서드를 모두 갖게 됩니다.

Zero.prototype.constructor = Zero;/\*Zero.prototype.constructor 프로퍼티를

Zero 함수로 설정합니다. 이는 Zero 함수로 생성된 객체들이 Zero 함수를 생성자로

 사용했음을 나타냅니다. 이 프로퍼티를 설정하지 않으면 기본값인 Human 함수가

 Zero 함수의 생성자로 지정되어 생성된 객체들이 Human 함수를 생성자로 사용한

 것으로 나타나게 됩니다.\*/

Zero.prototype.sayName = function(){

    console.log(this.firstName+''+this.lastName);

};

var oldZero = new Zero('human','Zero','Cho');

console.log(Human.isHuman(oldZero));

oldZero.breathe();

결과 :

PS E:\오전수업\node\ch2> node literal

true

haaaaaaaaaaam