INTERPARK

2017 NOV.

{ .JS }

JavaScript Lab

배열와 딕셔너리

JavaScript Lab 5

01. 열거법 (Enumerable: for..in, for)

> 열거를 위해
딕셔너리 for...in 반복문 호출 시
사용자 프로토타입 프로퍼티도
출력되는 오류발생(프로토타입 오염)

> 이를 방지하기
딕셔너리는 Object인스턴스를
사용하고 Object.prototype에
프로퍼티 추가 지양

```
1
2   var dict = { alice: 34, bob: 24, chris: 62 };
3   var people = [];
4   for(var name in dict){
5     people.push(name + ":" + dict[name]);
6   }
7   people; //["alice:34", "bob:34", "chris:62"]
8
9   //사용자 정의 Dictionary 생성
10  function UserDict() {}
11      UserDict.prototype.count = function(){
12          var i = 0;
13          for(var name in this){
14            i++;
15          }
16          return i;
17      }
18      UserDict.prototype.toString = function(){
19      return "[object UserDict]";
20  }
21
22  var ud = new UserDict();
23  ud.alice = 34;
24  ud.bob = 24;
25  ud.chris = 62;
26  ud.count(); //??
27
28  //Object.prototype에 property 추가 지양
29  Object.prototype.getUpper = function(key){
30  return this[key].toUpperCase();
31  }
32  people;
```

01. 열거법 (Enumerable: for..in, for)

> 생산성을 위해 Object.prototype에 프로퍼티 추가해야 한다면 우측 방법 고려

1>
var x = Object.create(null) 을 이용하여 프로토타입 없는 객체 생성 가능(ES5 ↑)

```
1
2   //1)
3   //Object.Prototype에 property 추가
4   Object.prototype.getUpper = function(key){
5     return this[key].toUpperCase();
6   }
7
8   //Prototype이 비어있는 객체 생성
9   var dict = Object.create(null);
10  Object.getPrototypeOf(dict) === null; //true
11
12  dict.alice = 34;
13  dict.bob = 24;
14  dict.chris = 64;
15
16  var people = [];
17  for(var name in dict){
18    people.push(name + ": " + dict[name]);
19  }
20
21  console.log(people);
22  //["alice: 34", "bob: 24", "chris: 62"]
23
24
25
26
27
28
29
30
31
32
```

01. 열거법 (Enumerable: for..in, for)

2>
Object.defineProperty 메소드로 프로토타입 프로퍼티 추가(ES5 ↑)
>> enumerable false지정 시 for...in 반복문 노출 X

```
1
2   //2)
3   Object.prototype.allKeys = function(){
4     var result = [];
5     for(var key in this){
6       result.push(key);
7     }
8     return result;
9   }; //<-- 세미콜론 삭제시?
10
11  ({alice: 34, bob: 24, chris:62 }).allKeys(); //?
12
13  Object.defineProperty(Object.prototype, "allKeys",{
14  value: function(){
15       var result = [];
16       for(var key in this){
17         result.push(key);
18       }
19       return result;
20  },
21  writable: true,
22  enumerable: false,
23  configurable: true
24  });
25
26  ({alice: 34, bob: 24, chris:62 }).allKeys();
27  //["alice", "bob", "chris"]
28
29
30
31
32
```

01. 열거법 (Enumerable: for..in, for)

3> hasOwnProperty 사용

hasOwnProperty를 재정의 할 경우 문제발생 -> {}.hasOwnProperty.call(dict,"toString") 으로 회피

```
1
2   Object.prototype.getUpper = function(key){
3     return this[key].toUpperCase();
4   }
5
6   var dict = { alice:34, bob:24, chris:62 };
7
8   var people = [];
9   for(var name in dict){
10    if(dict,hasOwnProperty(name))
11      people.push(name + ": " + dict[name]);
12  }
13
14  console.log(people);
15  //["alice: 34", "bob: 24", "chris: 62"]
16
17  //hasOwnProperty overriding
18  dict.hasOwnProperty = 10;
19  dict.hasOwnProperty("alice")
20  //Uncaught TypeError: dict.hasOwnProperty is not
21  function
22  [].hasOwnProperty.call(dict, "alice") //true;
23
24  //프로퍼티 탐색 상황도 고려
25  if("alice" in dict){
26    dict.alice = 24;
27  }
28
29  "alice" in dict //true;
30  "bob" in dict //true;
31  "chris" in dict //true;
32  "oleh" in dict //false;
```

01. 열거법 (Enumerable: for..in, for)

>
위 패턴을 딕션너리의 연산에 적용한 클래스로 추상화하여 사용
(기존 문법보다 견고하며 사용 편리성 유지)

```
1
2   Object.prototype.addOneYear = function(key){
3     return this[key] = this[key] + 1;
4   }
5
6   //사용자 정의 Dictonary 추상화
7   function Dict(elements){
8     this.elements = elements || {};
9   }
10  Dict.prototype.has = function(key){
11    return {}.hasOwnProperty.call(this.elements, key);
12  }
13  Dict.prototype.get = function(key){
14    return this.has(key) ? this.elements[key] :
15  undefined;
16  }
17  Dict.prototype.set = function(key, val){
18    this.elements[key] = val;
19  }
20  Dict.prototype.remove = function(key){
21    delete this.elements[key];
22  }
23
24  var dict = new Dict({alice:34, bob:24, chris:62});
25  console.log(dict.has("alice")); //true
26  console.log(dict.get("bob")); //24
27  console.log(dict.has("valueOf")); //false
28
29  console.log(dict.has("addOneYear")); //false
30  console.log(dict.elements.addOneYear("alice")); //35
31
32  console.log(dict.has("hasOwnProperty")); //false
```

01. 열거법 (Enumerable: for..in, for)

> 딕셔너리 객체의 for...in 열거는 순서를 보장하지 않음
(실행환경에 따라 객체를 열거하는 순서 달라질 수 있음)

> 순서가 정해진 컬렉션 열거 시 딕셔너리 대신 배열사용
(with for 반복문)

```
1
2   //딕셔너리 객체의 for...in 열거는 순서를 보장하지 않음
3   function report(highScores){
4        var result = "";
5        var i = 1;
6        for(var name in highScores){
7          result += i + ". " + name + ": " +
8          highScores[name] + "\n";
9          i++;
10       }
11       return result;
12  }
13
14  console.log(report(
15  {
16  eunhak: 100,
17  gusik: 90,
18  nbok: 80
19  }));
20
21  /*
22  순서의존 데이터 열거를 위해
23  {} -> [], for..in -> for 사용
24  */
25  function reportFor(highScores){
26       var result = "";
27       for(var i = 0, n = highScores.length; i < n;
28       i++){
29         var score = highScores[i];
30         result += (i + 1) + ". " + score.name + ": " +
31         score.points + "\n";
32       }
```

01. 열거법 (Enumerable: for..in, for)

> 배열 반복시에는 for..in 대신 for 사용
길이를 재계산하지 않기 위해
length결과 값을 지역변수에 저장

```
1
2   //배열반복시에는 for...in대신 for사용
3   //길이를 재계산하지 않기 위해 length 값을 지역변수에 저장
4
5   var scores = [100, 90, 90, 100];
6   var total = 0;
7   for(var score in scores){
8     total += score;
9   }
10
11  var mean = total / scores.length;
12  console.log(mean); //??
13
14  ////배열 반복시 for 사용///
15  var scores = [100, 90, 90, 100];
16  var total = 0;
17  for(var i = 0, n = scores.length; i < n; i++){
18    total += scores[i];
19  }
20
21  var mean = total / scores.length;
22  console.log(mean);
23
24
25
26
27
28
29
30
31
32
```

01. 열거법 (Enumerable: for..in, for)

> 반복문 대신 반복메소드 (forEach, map, filter...)사용
코드 가독성 올리고 실수 방지
Array.prototype.forEach, map, filter, some, every 등

```
1
2   //반복문 대신 반복메소드 사용
3   //코드 가독성 올리고 실수 방지
4
5   //반복문 종료조건에 대한 사소한 실수
6   for(var i = 0; i <= n; i++ ) {}
7   for(var i = 1; i < n; i++ ) {}
8   for(var i = n; i >= 0; i-- ) {}
9   for(var i = n - 1; i > 0; i-- ) {}
10
11  //ES5 제공 반복 메소드, Array.prototype.forEach, map,
12  filter, some, every
13  var trimmed = [];
14  for(var i = 0, n = input.length; i < n; i++){
15    trimmed.push(input[i].trim());
16  }
17
18  var trimmed = [];
19  input.forEach(function(s){
20    trimmed.push(s.trim());
21  });
22
23  var trimmed = input.map(function(s){
24    s.trim();
25  });
26
27  var filterd = items.filter(function(listing){
28    return item.price >= min && items.price <= max;
29  });
30  //사용자 정의 반복함수
31  function takeWhile(a, pred){
32      var result = [];
```

01. 열거법 (Enumerable: for..in, for)

> 다만, 반복문내에서 흐름제어(break, continue)필요할 경우, 전통방법 사용, 대안으로 some, every

```
1
2   //다만, 반복문내애서 흐름제어(break, continue)필요할 경
3   우 전통방법 사용, 대안으로 some, every
4
5   [1,10,100].some(function(x){ return x > 5; })
6   // true => 바로종료
7   [1,10,100].some(function(x){ return x > 6; })
8   // false
9   [1,2,3,4,5].every(function(x){ return x > 0; })
10  // true
11  [1,2,3,4,5].every(function(x){ return x > 3; })
12  // false => 바로종료
13
14  function takeWhile(a, pred){
15       var result = [];
16       a.every(function(x, i){
17            if(!pred(x)){
18              return false; //break
19            }
20            result[i] = x;
21            return true //continue;
22       });
23       return result;
24  }
25
26
27
28
29
30
31
32
```

02. 유사배열객체

> Array.prototype을 상속 하지 않지만
해당 메소드를 사용할 수 있는 객체

> 유사배열객체:
arguments,
documents.getElementsByTagName결과값

```
1
2   //arguments객체는 Array.prototype 상속 X ->
3   arguments.forEach 사용 X
4   function highlight(){
5       [].forEach.call(arguments, function(widget){
6           widget.setBackgorud("yellow");
7       });
8   }
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

02. 유사배열객체

> 유사배열객체도
Array.prototype 메소드 사용 권장

> 유사배열객체의 조건
(Array.prototype 메소드 호환 조건)

1) 정수형 length 프로퍼티
2)객체의 키 프로퍼티가 정수형 인덱스

```
1
2   //유사배열객체도 Array.prototype 메소드 사용권장
3
4   var arrayLike = {0: "a", 1: "b", 2: "c", length: 3}
5   var result = Array.prototype.map.call(arrayLike,
6   function(s){
7           return s.toUpperCase();
8   });
9
10  console.log(result); //["A, "B", "C"]
11
12  var result = Array.prototype.map.call("abc",
13  function(s){
14          return s.toUpperCase();
15  });
16
17  console.log(result); //["A, "B", "C"]
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

03. Array 생성자 대신 배열 리터럴 사용

> Array 생성자는
하나의 인자값이 숫자일때 다르게 동작

> 배열 리터럴 사용으로
의도치 않은 오류 사전차단
(Array.prototype 메소드 호환 조건)

```
1
2   //3. Array 생성자 대신 배열 리터럴 사용
3
4   //배열 리터럴
5   var a = [1, 2, 3, 4, 5];a
6
7   //배열 생성자
8   var a = new Array(1, 2, 3, 4, 5);
9
10  //1번째 생성자 사용으로 인한 오류
11  function f(Array){
12    return new Array(1, 2, 3, 5, 6)
13  }
14  f(String); //new String(1);
15
16  //2번째 생성자 사용으로 인한 오류
17  Array = String;
18  new Array(1, 2, 3, 4, 5); //new String(1);
19
20  //3번째 생성자 사용으로 인한 오류
21  var a = [7];
22  var b = new Array(7);
23  console.log(a);
24  console.log(b);
25
26  b.forEach(function(a){
27    console.log(b);
28  })
29
30
31
32
```

Fin