배열

배열 : 정렬된 값의 집합
원소 : 배열 안의 값
인덱스 : 배열의 위치번호

자바스크립트 배열

- 타입이 고정적이지 않음 (같은 배열에 있는 원소의 값이 서로 다를 수 있다)
- 32비트 인덱스를 사용한다
- 크기가 동적이다
- 배열을 생성하거나 크기가 변경되어 다시 할당을 하더라도 배열 크기를 다시 선언할 필요가 없다
- 모든 배열은 객체 (배열이 객체의 프로토타입으로부터 속성들을 상속 받은 객체이기 때문)

1. 배열 만들기

1.1 배열 리터럴

배열 리터럴 사용하기
- 대괄호 안에 쉼표로 구분해 나열 한 것
- 값으로는 상수 뿐 아니라 임의의 표현식도 사용할 수 있다
- 객체 리터럴 또는 다은 배열 리터럴을 포함할 수 있다
- 배열 리터럴에서 빠진 부분이 존재할 경우, 해당 부분의 원소 값은 undefined가 된다
- 제일 마지막에 쉼표를 추가할 수 있다
- ex) var arr = [2, 3, 4, 5, , true, "a",];

1.2 Array() 생성자

Array() 생성자 사용하기
- 인자 없이 호출 (아무 원소도 없는 빈 배열, 리터럴과 동일)

```
var a = new Array();
```

- 길이를 의미 하는 숫자 값을 인자로 주어 호출 (배열의 값과 인덱스 값이 존재하지 않는다)

```
var a = new Array(44);
```

- 두 개 이상의 원소 값 or 숫자가 아닌 원소 값 하나를 명시적으로 지정

```
var a = new Array(5, 4, 3, 2, 1, "test, test");
```

배열의 생성자를 사용하는 것 보다 리터럴이 훨 씬 간단하다
크기를 지정해주지 않으면 최대 크기로 잡는다 (2^{32}-1)

2. 배열의 원소 읽고 쓰기

배열의 각 원소에 접근 할 시 [ ] 연산자 사용
- 참조변수(배열 명)[음이 아닌 정수 값으로 평가되는 임의의 표현 식(인덱스 값)]

```
val a = arr[4], var a = ["test"], a[a[i]]= a[0]
```

자바스크립트는 사용자가 명시한 숫자 배열 인덱스를 문자열 형태로 바꿔서 속성 이름으로 사용
배열의 인덱스와 객체의 속성이름을 올바르게 구별하는 것이 유용하다
- 모든 인덱스의 값은 속성 이름 , 0과 2^{32}-1 사이의 정수 값 속성이름만 인덱스

배열은 필요한 경우 length 속성 값을 자동으로 갱신한다

3. 희소배열

배열에 속한 원소의 위치가 연속적이지 않은 배열
length 속성의 값은 원소의 개수보다 항상 큼
보통 배열보다 일반적으로 느리고, 메모리를 많이 사용하고, 원소를 찾는 시간이 오래 걸림

4. 배열의 길이

모든 배열엔 length속성이 있다. (객체와 배열을 구분하는 중요한 특징)
배열에 배열의 크기와 같거나 큰 값으로 인덱스 값을 설정 할 경우 length = 인덱스 값 +1 이다
배열은 항상 length 값과 같게 구현되어 있다
- var test = {1,2,3,4,5}; 에서 test.length = 0; 을 하면 모든 element를 삭제 하고 결과는 { } 이다

5. 배열에 원소를 추가하거나 삭제하기

5.1 추가

직접 할당하기 (인덱스를 지정해 할당)
push() 메서드 사용하기 (배열의 끝에 원소를 추가)
unshift() 메서드 사용하기 (배열의 맨 앞에 원소를 추가, 기존원소들의 인덱스 값이 1씩 커짐)

5.2 삭제

delete 로 삭제해도 배열의 길이는 변하지 않음
- undefined 를 할당하는 것과 같음

length 속성으로 배열의 끝에서부터 원소삭제가능
pop() 메서드 사용하기 (배열의 length를 하나 줄이고 삭제된 값을 반환)
shift() 메서드 사용하기 (배열의 맨 앞에 원소를 삭제, 모든 원소의 인덱스 값을 하나씩 감소)
* splice() 메서드는 추가, 삭제, 대치하는 범용 메서드이다

6. 배열 순회하기

가장 기본적인 방법은 for 문을 사용하는 것이다
배열의 원소를 사용하기 전에 null , undefined, 빈 원소가 있는지 확인 하고 제외 시켜줘야 한다
(배열의 length를 한번만 가져와서 사용하도록 루프를 최적화 시키는 경우가 있기 때문이다)

```
for(var i = 0, len = keys.length; i < len; i++){
        if(!a[i])
                continue;  // null , undefined, 빈 원소일 때 건너뜀
}
```

for / in 루프는 상속받은 속성 이름까지 넘겨준다
(원하지 않은 속성을 제외하고 추가된 속성만 테스트하는 경우가 아니라면 사용하지 말아야 한다)

7. 다차원 배열

다차원 배열을 지원하지는 않지만 배열의 배열을 사용해 비슷하게 사용 할 수 있다
배열 내의 배열에 있는 원소에 접근하는 법
- [ ]연산자를 두 번 사용하면 된다
다차원 배열 만드는 법

```
var arr = new Array(10);              // 10개의 행 만들기
for(var i = 0; I < arr.length; i++){
        arr[i] = new Array(10);       // 각 행에 10개의 열을 만들기
}
```

8. 배열 메서드

ECMAScript 3의 Array.prototype 에서 배열을 다루는 데 필요한 여러 종류의 함수들을 정의함

8.1 join()

배열의 모든 원소들을 문자열로 변환하고 그 문자들을 이어 붙인 결과를 반환
구분자는 쉼표(,)가 기본 값으로 사용
String.split() 메서드와 반대로 작동

```
var a = [1,2,3];
a.join(); // "1,2,3"
a.join(" "); //"1 2 3"
a.join("") //"123"
var b =new Array(10); //길이가 10인 배열
b.join('-') // --------- 비어있으므로 사이의 9개의 -의 문자열이 생김
```

8.2 reverse()

배열의 원소 순서를 반대로 정렬하여 반환
배열 안에서 직접 수행된다

```
var a = [1,2,3];
a.revarse() // a는 [3,2,1]이 된다
```

8.3 sort()

배열 안의 원소들을 정렬하여 반환
별도의 전달인자가 없을 경우 배열 안의 원소들을 알파벳순으로 정렬
대소 문자 구별 없이 정렬 하고 싶으면 toLowerCase() 메서드를 사용

```
var a = new Array("banana", "cherry", "apple");
a.sort(); // 알파벳순으로 정렬
var b = a.join(", "); //"apple, banana, cherry"
var c = [33,4,111,222];
c.sort(function(a,b){ //오름차순 : 4, 33, 222, 1111
        return a-b;
});
c.sort(function(a,b){ //내림차순 : 1111,222,33,4
        return b-a;
});
```

8.4 concat()

기존 배열의 모든 원소에 concat() 메서드의 전달인자들을 추가한 새로운 배열 반환
배열의 원소 중에 배열이 있는 중접 배열의 경우 원소를 꺼내지 않음

```
var a = [1,2,3];
a.concat(4,5);//[1,2,3,4,5]
a.concat([4,5]);//[1,2,3,4,5]
a.concat([4,5],[6]);//[1,2,3,4,5,6]
a.concat(4,[5,[6]]);//[1,2,3,4,5[6]]
```

8.5 slice()

부분 배열을 반환 (배열에서 잘라낸 원소들을 담은 새 배열)
처음과 끝을 알리는 전달인자 2개를 받음
전달인자가 음수면 우측(마지막 원소)으로 부터 앞쪽으로 전달인자 만큼 위치한 원소를 가리킴

```
var a = [1,2,3,4,5,6,7]
a.slice(0,4); //[1,2,3,4]
a.slice(4); //[5,6,7]
a.slice(0,-2); //[1,2,3,4,5]
a.slice(-4,-2); //[4,5]
```

8.6 splice()

배열의 원소를 삽입, 제거 시 범용으로 사용 가능
처음과 끝을 알리는 전달인자 2개를 받음 3번째 전달인자 부터는 새롭게 삽입할 원소들을 지정
호출 시 배열 바로 수정

(처음, 처음으로부터 얼마나삭제할것인지, 새로삽입할 원소, 새로삽입할 원소,...)

삭제한 배열을 반환한다

```
var a = [1,2,3,4,5,6,7,8];
a.splice(4); //[5,6,7,8]반환 a는 [1,2,3,4]
a.splice(1,2); //[2,3]반환 a는 [1,4]
a.splice(1,1); //[4]반환 a는 [1]
var b = [1,2,3,4,5];
b.splice(2,0,'a','b'); //[]반환 b는 [1,2,'a','b',3,4,5]
b.splice(2,2,[1,2],3); //['a','b']반환 b는 [1,2,[1,2],3,3,4,5]
```

8.7 push(), pop()

배열의 맨뒤에 원소를 추가, 제거하고 배열을 스택처럼 조작 가능 (FILO 스택을 구현할 수 있음)
var stack =[];
stack.push(1,20); //[1,2] 2반환
stack.pop(); //[1] 1반환
stack.push(1,20); //[1,1,20] 3반환
stack.pop(); //[1,1] 2반환
stack.pop() //[1] 1반환

8.8 unshift(), shift()

배열의 맨 앞에 원소를 추가, 제거하고 그에 따른 새로운 length 값을 반환한다
var a= [];
a.unshift(1); //[1] 1반환
a.unshift(22,333); //[1,22,333] 3반환
a.shift(); //[1,22] 333반환
a.unshift(4,[5,6]); //[1,22,4,[5,6]] 4반환
a.shift(); //[1,22,4] [5,6]반환
a.shift(); //[1,22] 4반환

8.9 toString() toLocationString()

toString 메서드는 배열의 모든 원소를 문자열로 변환하고 쉼표(,)로 분리한 목록을 반환한다
변환된 문자열은 구분자를 포함하지 않는다
toLocationString()은 toString의 지역화 버전
[1, 2, 3].toString(); //결과 '1, 2, 3'
["a", "b", "c"].toString(); //결과 'a, b, c'
[1,['a','b'], 2].toString; //결과 '1,a,b,2'

9. ECMAScript 5 배열 메서드

배열을 순회, 매핑, 필터링, 테스팅, 감소, 검색하기 위해 9가지의 새로운 메소드를 정의했다
첫번째 전달인자로 함수를 받는다 단 희소배열이라면 빈 원소의 경우 함수 호출을 하지 않는다
대부분 첫 번째 전달인자로 지정한 함수는 세 개의 전달인자를 가지고 호출된다
(배열 원소의 값, 인덱스, 배열 그 자체)
첫 번째 전달인자인 함수는 마치 두 번째 인자의 메서드처럼 호출된다
(두 번째 인자는 첫 번째 전달인자인 함수 안에서 this 키워드의 값으로 사용된다)

9.1 forEach()

배열을 순회하는 메서드
각 원소에 대하여 지정한 함수를 각각 호출한다
break 문으로 종료 시킬 수 없다
var data = [1,2,3,4,5];
//배열에 속한 모든 원소의 합을 계산한다
var sum = 0;
data.forEach(function(value) {sum += value; }); //각 원소의 값을 sum에 더한다.
sum //15
//각 원소의 값을 증가시킨다
data.forEach(function(v, 1, a) { a[i] = v+ 1; });
data//[2,3,4,5,6]

9.2 map()

배열의 각 원소를 메서드의 첫 번째 전달인자로 지정한 함수에 전달하고, 해당 함수의 반환 값을 새로운 배열로 반환
희소배열이라면 희소배열을 반환
a = [1,2,3];
b = a.map(function(x) {resturn x*x]); //b는 [1,4,9]

9.3 filter()

배열의 일부분을 반환한다(단, 전달한 함수는 조건자 함수(true false)여야 한다)
반환값이 true, true인 조건식이면 조건자 함수를 통과해 메서드가 반환할 배열에 추가가 된다
희소배열의 경우엔 빈 원소를 건너뛰어서 반환되는 배열에는 빈 원소가 없다
a = [5,4,3,2,1];
filtertest = a.filter(function(x) {return x <4]); //[1,2,3]
//희소배열의 빈 원소 제거하는 법
var danse = sparse.filter(function() { return true; });
//빈 원소의 간격을 좁히고 undefined, null 값을 갖는 원소도 함께 제거하는 법
a=a.filter(function(x) { return x !== undedined && != null; });

9.4 every(), some()

배열을 단정한다 (배열의 각 원소에 대하여 true or false를 반환하는 함수)
every() : 배열의 모든 원소에 대하여 true를 반환하는 경우에 true 반환
some() : 일부 원소에 대하여 true인 경우 true를 반환
반환 값이 결정되면 배열의 원소순회를 중단한다
빈 배열인 경우 every()는 항상 true, some()는 항상 false를 반환
a = [1,2,3,4,5];
a.every(function(x) {return x<10; }) //true
a.every(function(x) {return x%2 ===0; }) //false
b = [1,2,3,4,5];
b.some(function(x) {return x%2 ===0; }) //true
b.some(isNaN) //false

9.5 reduce(), reduceRight()

조건함수를 사용하여 배열의 원소들을 하나의 값으로 결합
결합방식은 inject와 fold 로 알려져 있다
두 개의 인자를 갖는데 첫 번째는 배열원소의 감소작업을 하는 함수 두 번째는 시작 값(생략가능)
reduce()는 오름차순, reduceRight()는 내림차순으로 처리한다
숫자만이 아닌 두 객체가 공통으로 가진 속성들을 새 객체에 담아 반환한다 (merge 같은 기능)

```
var a = [1,2,3,4,5]
var sum = a.reduce (function(x,y) { return x+y }, 0);          //배열 a의 원소들의 합
var sum = a.reduce (function(x,y) { return x*y }, 0);          //배열 a의 원소들의 곱
var sum = a.reduce (function(x,y) { return (x>y)?x:y; }, 0);  //배열 a의 원소 중 가장 큰 값
//거듭제곱계산 (오른쪽에서 왼쪽으로 진행)
var b = [2,3,4]
var big = a.reduceRight(function(accumlator,value) {
        return Math.pow(value,accumulator);
});
```

9.6 indexOf(), lastIndexOf()

배열의 원소 중에 특정한 값을 찾는다.
찾으면 해당 인덱스를 찾지 못하면 -1을 반환한다
함수를 인자로 받지 않고 첫째 인자는 찾고자 하는 값 두 번째 인자는 찾고자 하는 범위이다
lastIndexOf는 배열의 마지막 원소부터 검색을 하고 음수 값은 배열의 상대적인 위치이다

```
a = [0,1,2,1,0];
a.indexOf(1) //반환값 1  a[1]은 1
a.lastIndexOf(1) //반환값 3 a[3]은 1이다
a.indexOf(3) //반환값 -1 값이 0인 원소는 없음
```

10. 배열타입

ECMAScript5에서는 Array.isArray() 함수를 통해 배열인지 판단 할 수 있다
ECMAScript5 이전에는 instanceof 를 사용시엔 종종 문제점이 웹 브라우저에서 발생한다
- 하나의 창 또는 프레임이 열렸을 때 발생

```
Array.isArray([])//true
Array.isArray({})//false
```

11. 유사 배열 객체

Arguments
클라이언트 자바스크립트에서는 상당수의 DOM 메서드가 배열과 유사한 객체를 반환한다
배열 메서드를 범용으로 정의한 이유는 배열과 유사한 객체에서도 작동하게 하기 위해서다

12. 문자열을 배열처럼 사용하기

대부분의 최근 브라우저에서는 문자열을 읽기 전용 배열처럼 다룬다([ ] 를 이용해 접근 가능)
문자열에 범용 배열 메서드를 바로 사용할 수 있다
문자열은 변하지 않는 값이라서 push(), sort(), reverse(), splice() 는 적용되지 않는다