

제03장

연산자

구디아카데미 ▷ 민경태 강사

```
each: function(e, t, n) {  
  var r, i = 0,  
      o = e.length,  
      a = M(e);  
  if (n) {  
    if (a) {  
      for (; o > i; i++)  
        if (r = t.apply(e[i], n), r ===  
    } else  
      for (i in e)  
        if (r = t.apply(e[i], n), r ===  
  } else if (a) {  
    for (; o > i; i++)  
      if (r = t.call(e[i], i, e[i]))  
    } else  
      for (i in e)  
        if (r = t.call(e[i], i, e[i]))  
    return e  
  },  
  trim: b && !b.call("\uffff\u00a0") ?  
    return null == e ? "" : b.call(  
  } : function(e) {  
    return null == e ? "" : (e +  
  },  
  makeArray: function(e, t) {  
    var n = t || [];  
    return null != e && (M(Obj  
  },  
  isArray: function(e, t, n) {  
    var r;  
    if (t) {  
      if (n) return n.c  
      for (n = t.length;  
        if (n in t)  
    }  
  }
```

학습목표

1. 산술 연산자에 대해서 알 수 있다.
2. 대입 연산자에 대해서 알 수 있다.
3. 증감 연산자에 대해서 알 수 있다.
4. 비교 연산자에 대해서 알 수 있다.
5. 논리 연산자에 대해서 알 수 있다.
6. 조건 연산자에 대해서 알 수 있다.
7. 문자열 연산자에 대해서 알 수 있다.
8. 연산자 우선순위에 대해서 알 수 있다.

```
each: function(e, t, n) {  
  var r, i = 0,  
      o = e.length,  
      a = M(e);  
  if (n) {  
    if (a) {  
      for (; o > i; i++)  
        if (r = t.apply(e[i], n), r ===  
    } else  
      for (i in e)  
        if (r = t.apply(e[i], n), r ===  
  } else if (a) {  
    for (; o > i; i++)  
      if (r = t.call(e[i], i, e[i])  
  } else  
    for (i in e)  
      if (r = t.call(e[i], i, e[i])  
  return e  
},  
trim: b && !b.call("\uffff\u00a0") ?  
  return null == e ? "" : b.call(  
} : function(e) {  
  return null == e ? "" : (e + "  
},  
makeArray: function(e, t) {  
  var n = t || [];  
  return null != e && (M(Ob  
}),  
isArray: function(e, t, n) {  
  var r;  
  if (t) {  
    if (n) return m.c  
    for (r = t.length  
    if (n in t  
  }  
}
```

목차

1. 산술 연산자
2. 대입 연산자
3. 증감 연산자
4. 비교 연산자
5. 논리 연산자
6. 조건 연산자
7. 문자열 연산자
8. 연산자 우선순위

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > r ? Math.max(0, r + n) : r; r--;)
            if (n in t && t[n] === e) return n;
    }
}

```

1. 산술 연산자

산술 연산자

■ 정수 연산

+ 더하기

- 빼기

* 곱하기

/ 몫

% 나머지

■ 실수 연산

+ 더하기

- 빼기

* 곱하기

/ 나누기

% 나머지

동일한 연산자이지만
자료형에 따라 연산 결과에 차이가 있다.

정수 연산

■ 정수 연산의 예시

연산 결과

$5 + 3$ ► 8

$5 - 3$ ► 2

$5 * 3$ ► 15

$5 / 3$ ► 1

$5 \% 3$ ► 2



$$\begin{array}{r} 1 \text{ 몫 } / \\ 3 \overline{) 5} \\ \underline{- 3} \\ 2 \text{ 나머지 } \% \end{array}$$

실수 연산

■ 실수 연산의 예시

연산	결과
$3.0 + 1.2$	▶ 4.2
$3.0 - 1.2$	▶ 1.8
$3.0 * 1.2$	▶ 3.6
$3.0 / 1.2$	▶ 2.5
$3.0 \% 1.2$	▶ 0.6

참고) 실수 연산의 경우 소수점 단위를 근사값으로 처리하기 때문에 약간의 오차가 발생할 수 있다.

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r < n; r++)
            if (n in t && t[r] === e) return r;
    }
}

```

2. 대입 연산자

대입 연산자

- 대입 연산자 : 등호(=)
- 등호(=)의 오른쪽 값을 왼쪽에 있는 변수로 전달함

```
int a;  
int b;
```

```
a = 5;  
b = a;
```

5을 변수 a로 전달한다.

변수 a의 값을 변수 b로 전달한다.

복합 대입 연산자

■ 연산과 대입을 동시에 진행하는 연산자

- += 더한 결과를 대입
- -= 뺀 결과를 대입
- *= 곱한 결과를 대입
- /= 나눈 몫을 대입
- %= 나눈 나머지를 대입

복합 대입 연산자

대입 연산

```
int a = 0;
```

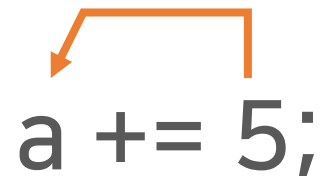


```
a = a + 5;
```

변수 a에 5를 더한 결과를 변수 a에 전달한다.

복합 대입 연산

```
int a = 0;
```



```
a += 5;
```

a = a + 5와 같은 결과를 도출한다.

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r in t && t[r] === e) return r;
    }
}

```

3. 증감 연산자

증감 연산자

■ ++ : 1 증가 연산

- ++a 변수 a값을 1 증가시킨 뒤 사용함 (전위 연산)
- a++ 변수 a값을 사용한 뒤 1 증가시킴 (후위 연산)

■ -- : 1 감소 연산

- --a 변수 a값을 1 감소시킨 뒤 사용함 (전위 연산)
- a-- 변수 a값을 사용한 뒤 1 감소시킴 (후위 연산)

전위 연산

전위 연산

```
int a = 0;  
int b;
```

```
  ↙  
b = ++a;
```

변수 a를 1 증가시킨 뒤 변수 b에 전달하므로
변수 a와 변수 b는 모두 1 값을 가진다.

후위 연산

후위 연산

```
int a = 0;
```

```
int b;
```

```
b = a++;
```



변수 a를 변수 b에 전달한 뒤 변수 a를 1 증가시키므로
변수 a는 1, 변수 b는 0 값을 가진다.

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r in t && t[r] === e) return r;
    }
}

```

4. 비교 연산자

비교 연산자

- 크기를 비교한 뒤 boolean 자료형의 결과(true, false)를 반환함

- 종류

- $a > b$ a가 b보다 크면 true, 아니면 false 반환
- $a \geq b$ a가 b보다 크거나 같으면 true, 아니면 false 반환
- $a < b$ a가 b보다 작으면 true, 아니면 false 반환
- $a \leq b$ a가 b보다 작거나 같으면 true, 아니면 false 반환
- $a == b$ a와 b가 같으면 true, 아니면 false 반환
- $a != b$ a와 b가 다르면 true, 아니면 false 반환

문자 비교

- 문자들은 고유의 코드값(정수값)을 가지고 있음
- 문자들을 비교 연산자로 비교하면 코드값을 비교한 결과를 반환함
 - 'A'는 'B'보다 작다.
 - '0'은 '1'보다 작다.
 - 'a'는 'b'보다 작다.
- 주요 문자들의 코드값

구분	문자	코드값(10진수)
아라비아 숫자	'0', '1', '2', ... '9'	48, 49, 50, ... 57
영문 대문자	'A', 'B', 'C', ... 'Z'	65, 66, 67, ... 90
영문 소문자	'a', 'b', 'c', ... 'z'	97, 98, 99, ... 122

대입 연산과 동등 비교 연산 주의

- 대입 연산(=)와 동등 비교 연산(==) 혼동 주의

```
int a;
```

```
a = 10;
```

변수 a에 10을 전달하시오.

```
int a;
```

```
a == 10;
```

변수 a가 10과 같으면 true
아니면 false를 반환하시오.

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > r ? Math.max(0, r + n) : r; n in t && t[n] === e) return n;
    }
}

```

5. 논리 연산자

논리 연산자

- &&

논리 AND

두 값이 모두 true이면 true, 아니면 false 반환

- ||

논리 OR

두 값 중 하나라도 true이면 true, 아니면 false 반환

- !

논리 NOT

전달된 값의 반대 값을 반환

논리 AND

- &&

두 값이 모두 true이면 true 반환, 하나라도 false이면 false 반환

```
int a = 10;  
int b = 10;  
boolean c;
```

```
c = a == 10 && b == 10 ;
```

두 값이 모두 true이므로 true를 반환한다.

true

true

논리 OR

- ||

두 값 중 하나라도 true이면 true 반환, 모두 false이면 false 반환

```
int a = 10;  
int b = 10;  
boolean c;
```

```
c = a == 10 || b == 20 ;
```

true

false

true가 포함되어 있으므로 true를 반환한다.

논리 NOT

- !

전달된 값이 true이면 false 반환, 전달된 값이 false이면 true 반환

```
int a = 10;  
boolean c;
```

```
c = ! a == 10;
```

true

true가 전달되었으므로 false를 반환한다.

숏 서킷

- Short Circuit
- 논리 연산의 최종 결과가 결정되면 남아 있는 연산은 모두 생략함
- 논리 AND
 - 연산 중 false가 발생하면 최종 결과는 false이므로 논리 AND 연산이 종료됨
- 논리 OR
 - 연산 중 true가 발생하면 최종 결과는 true이므로 논리 OR 연산이 종료됨

숏 서킷 예시

- 변수 a, b에 저장된 값은?

```
int a = 10;  
boolean b;
```

```
b = a > 10 && ++a > 10 ;
```

① false 발생

② false 발생시

최종 결과는 false로 결정되므로 이후 연산을 무시한다.

③ 실행을 안한다.

최종적으로 변수 a는 10, 변수 b는 false를 가진다.

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r in t && t[r] === e) return r;
    }
}

```

6. 조건 연산자

조건 연산자

- 1항에 boolean 값을 반환하는 조건식을 사용함
- 1항의 결과가 true이면 2항의 값을 반환함
- 1항의 결과가 false이면 3항의 값을 반환함
- 자바에서 유일하게 3항을 사용하기 때문에 삼항 연산자라고도 부름
- 형식
조건식 ? 조건식이 true인 경우 : 조건식이 false인 경우
1항 2항 3항

조건 연산자 예시

- 두 값을 비교해서 큰 값을 저장하는 코드

```
int a = 10;
```

```
int b = 20;
```

```
int max;
```

```
max = a >= b ? a : b ;
```

1항 2항 3항

1항의 결과가 true이면 2항이 전달되고,
1항의 결과가 false이면 3항이 전달된다.

```

each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break;
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break;
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break;
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e);
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n; n in t && t[n] === e) return n;
  }
}

```

7. 문자열 연산자

문자열 연산자

- + : 문자열 연결 연산자
- 문자열이 포함된 + 연산은 문자열을 2개 항을 연결한 문자열을 반환함

```
int a = 10;  
String b;
```

```
b = a + "살";
```



b에는 "10살"이 저장된다.

숫자를 문자열로 변환하기

- 숫자에 빈 문자열("")을 더하면 문자열이 됨

```
String str;
```

```
str = "" + 10;
```



str에는 문자열 "10"이 저장된다.


```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r < n; r++)
            if (n in t && t[r] === e) return r;
    }
}

```

8. 연산자 우선순위

연산자 우선순위

- 동일한 우선순위를 가진 연산의 경우 왼쪽에서 오른쪽으로 이동하면서 먼저 나타난 연산을 수행함
- 우선순위가 낮은 연산을 먼저 수행하려면 괄호()로 해당 연산을 묶어서 처리함

연산자 우선순위

우선순위 높음



(), .

expr++, expr--

+ (양수 부호), - (음수 부호), !, ++expr, --expr

강제 형 변환(Casting)

*, /, %

+(덧셈), -(뺄셈)

<<, >>, >>>

<, >, <=, >=, instanceof

==, !=

& (비트 AND)

^ (비트 XOR)

| (비트 OR)

&& (논리 AND)

|| (논리 OR)

? : (조건 연산)

우선순위 낮음



=, +=, -=, *=, /=, %= (복합 대입)