

# 횡단 탐색(Traversing)



## ■ 선택자를 사용한 Selection

- HTML 요소를 찾는 범위(context)는 기본적으로 HTML 문서 단위
- HTML 문서의 크기가 크다면, HTML 문서 전체에서 찾는 작업을 여러번 반복할 경우 성능에 좋지 않은 영향을 준다.

## ■ 횡단 탐색 메서드

- 기존 선택된 요소들을 기점으로 해서 다른 요소를 선택하는 기능을 제공'
- 선택자만을 사용했을 때보다 선택/조회 성능 대폭 향상

## ■ "문서 전체에서 찾는 작업의 횟수를 최소화하자"

# 필터링 메서드(1)



## ■ 필터링 메서드란?

- 이미 선택한 요소에 추가적인 조건을 부여해 걸러내는 메서드
- 논리식으로 보자면 AND 조합임

[ 표 08-01 : 필터링 메서드 ]

메서드	설명
first( )	기존 선택된 요소 중 첫 번째 요소를 선택한다.
last( )	기존 선택된 요소 중 마지막 요소를 선택한다.
eq( )	기존 선택된 요소 중 특정 인덱스 위치에 있는 요소를 선택한다.
filter( )	기존 선택된 요소에서 지정한 선택자 조건을 만족하는 요소를 선택한다.
has( )	기존 선택된 요소의 후손 중에서 지정한 선택자 조건을 만족하는 요소를 선택한다.
not( )	기존 선택된 요소에서 지정한 선택자 조건을 만족하는 요소를 제거한 다음 나머지 요소를 선택한다. filter( )와 반대 기능을 제공한다.
slice( )	기존 선택된 요소의 시작 위치부터 마지막 위치까지 지정해 일부 요소를 선택한다.
map( )	map( ) 메서드의 인자로 전달된 익명 함수(anonymous function)에 선택된 요소를 하나씩 전달하여 반복 실행한 뒤 jQuery 객체를 리턴한다.
is( )	선택된 요소가 지정된 선택자로 선택될 수 있는 요소인지를 true/false로 리턴한다.

## 필터링 메서드(2)



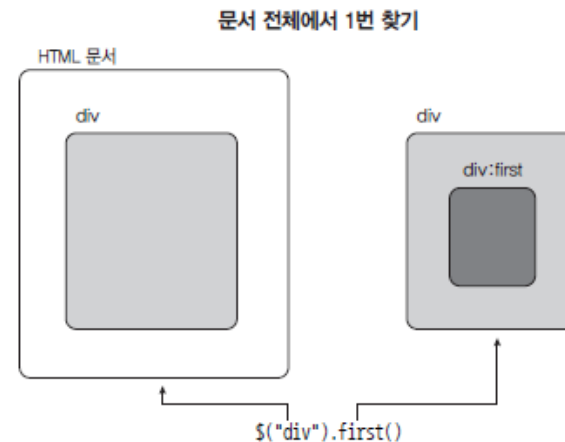
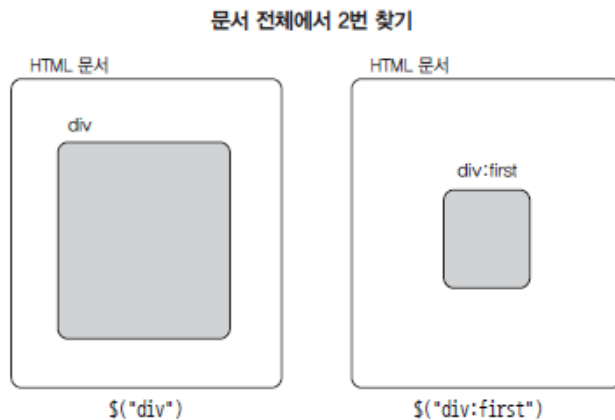
### ■ first(), last(), eq() 메서드

- 이 코드는 문서 전체에서 찾는 작업을 두 번 수행!!

```
$("#div").css("color", "blue");  
$("#div:first").css("font-style", "italic");
```

- 아래 코드는 이전 코드와 최종 결과는 같지만 문서 전체에서 찾는 작업은 한번만 수행

```
$("#div").css("color", "blue").first().css("font-style", "italic");
```



# 필터링 메서드(3)



## 예제 08-02

```
13: <body>
14:   <div id="content">
15:     <div id="a">김수한무</div>
16:     <div>거북이와 두루미</div>
17:     <div class="test">삼천갑자 동방삭</div>
18:     <div class="test main">치치카포 사리사리센타</div>
19:     <div>위리위리 세브리깡</div>
20:     <div>무두셀라 구름이</div>
21:     <div id="b">허리케인에 담벼락</div>
22:     <div class="main">담벼락에 서생원</div>
23:     <div class="test">서생원에 고양이</div>
24:     <div id="a">고양이에 바둑이</div>
25:     <div>바둑이는 돌돌이</div>
26:   </div>
27: </body>
```

[예제 08-02]

```
01: <script type="text/javascript">
02: $(document).ready(function() {
03:   var divs = $("#content > div").css("color", "blue");
04:   var first = divs.first().css("font-size", "20pt");
05:   var eq2 = divs.eq(2).css("border", "solid 1px black");
06:   console.dir(divs);
07:   console.dir(first);
08:   console.dir(eq2);
09: });
10: </script>
```



김수한무

거북이와 두루미

삼천갑자 동방삭

치치카포 사리사리센타

위리위리 세브리깡

무두셀라 구름이

허리케인에 담벼락

허리케인에 담벼락

담벼락에 서생원

서생원에 고양이

고양이에 바둑이

바둑이는 돌돌이

# 필터링 메서드(4)



- 예제 08-02 실행 결과
  - 기존 요소를 바탕으로 필터링한 jQuery객체는 prevObject라는 속성 존재
  - prevObject는 이전에 선택된 jQuery 객체를 가리킴

```
▼ jQuery.fn.init[12] ⓘ  
▶ 0: div#a  
▶ 1: div  
▶ 2: div.test  
▶ 3: div.test.main  
▶ 4: div  
▶ 5: div  
▶ 6: div#b  
▶ 7: div.test  
▶ 8: div.main  
▶ 9: div.test  
▶ 10: div#a  
▶ 11: div  
length: 12  
▶ prevObject: jQuery.fn.init[1]  
▶ __proto__: Object[0]
```

```
▼ jQuery.fn.init[1] ⓘ  
▶ 0: div#a  
length: 1  
▶ prevObject: jQuery.fn.init[12]  
▶ __proto__: Object[0]
```

```
▼ jQuery.fn.init[1] ⓘ  
▶ 0: div.test  
length: 1  
▶ prevObject: jQuery.fn.init[12]  
▶ __proto__: Object[0]
```

# 필터링 메서드(5)



## ■ filter( ), not( ) 메서드

- 이 코드는 문서 전체에서 찾는 작업을 두 번 수행!!

```
$("#div").css("color", "blue");  
$("#div.test").css("font-style", "italic");
```

- 아래 코드는 이전 코드와 최종 결과는 같지만 문서 전체에서 찾는 작업은 한번만 수행

```
$("#div").css("color", "blue").filter(".test").css("font-style", "italic");
```

[예제 08-03 : filter()]

```
01: <script type="text/javascript">  
02: $(document).ready(function() {  
03:   var divs = $("#content > div").css("color", "blue");  
04:   var tests = divs.filter(".test").css("font-size", "20pt");  
05:   console.log(tests);  
06:   console.log($("#content > div.test"));  
07:  
08:   var nots = divs.not(".test").css("border", "solid 1px black");  
09: });  
10: </script>
```

# 필터링 메서드(6)



## 예제 08-03 실행 결과

김수한무
거북이와 두루미
삼천갑자 동방삭
치치카포 사리사리센타
워리워리 세브리깁
무두셀라 구름이
허리케인에 담벼락
허리케인에 담벼락
서생원에 고양이
고양이에 바둑이
바둑이는 돌돌이

08-03.html:11

```
jQuery.fn.init[4]
  0: div.test
  1: div.test.main
  2: div.test
  3: div.test
  length: 4
  prevObject: jQuery.fn.init[11]
  __proto__: Object[0]
```

08-03.html:12

```
jQuery.fn.init[4]
  0: div.test
  1: div.test.main
  2: div.test
  3: div.test
  length: 4
  prevObject: jQuery.fn.init[1]
  __proto__: Object[0]
```

# 필터링 메서드(7)



## ■ has() 메서드

- 기존 선택된 요소 중에서 지정된 선택자로 선택할 수 있는 자식 요소를 포함한 요소로 필터링

```
07: <script type="text/javascript">
08: $(document).ready(function() {
09:   $("div").has("h1").css("font-style", "italic");
10: });
11: </script>
12: </head>
13: <body>
14:   <div><h1>Item 1</h1></div>
15:   <div><p>Item 2-1</p><p>Item 2-2</p></div>
16:   <div><h1>Item 3</h1></div>
17:   <div><div>Item 4-1</div><div>Item 4-2</div></div>
18: </body>
```



***Item 1***

Item 2-1

Item 2-2

***Item 3***

Item 4-1

Item 4-2



# 필터링 메서드(8)



## ■ slice() 메서드

- 전체 요소 중에서 위치 정보를 이용해 일부 요소를 추출하는 메서드

```
<body>
  <div id="a1">Item 1</div>
  <div id="a2">Item 2</div>
  <div id="a3">Item 3</div>
  <div id="a4">Item 4</div>
  <div id="a5">Item 5</div>
  <div id="a6">Item 6</div>
  <div id="a7">Item 7</div>
  <div id="a8">Item 8</div>
</body>
```

```
> divs.slice(4)
< [ <div id="a5">Item 5</div>, <div id="a6">Item 6</div>,
    <div id="a7">Item 7</div>, <div id="a8">Item 8</div>]
> divs.slice(1,3)
< [ <div id="a2">Item 2</div>, <div id="a3">Item 3</div>]
> divs.slice(4,-2)
< [ <div id="a5">Item 5</div>, <div id="a6">Item 6</div>]
> divs.slice(-3,-1)
< [ <div id="a6">Item 6</div>, <div id="a7">Item 7</div>]
```

# 필터링 메서드(9)



## ■ is(), map() 메서드

- is() : 기존의 선택된 요소 중 지정된 선택자로 선택되는 요소가 존재하는지를 true/false로 리턴

```
<body>
  <div id="content">
    <div id="a">김수한무</div>
    <div>거북이와 두루미</div>
    <div class="test">삼천갑자 동방삭</div>
    <div class="test main">치치카포 사리사리센타</div>
    <div>워리워리 세브리깁</div>
    <div>무두셀라 구름이</div>
    <div id="b">허리케인에 담벼락</div>
    <div class="main">담벼락에 서생원</div>
    <div class="test">서생원에 고양이</div>
    <div id="a">고양이에 바둑이</div>
    <div>바둑이는 돌돌이</div>
  </div>
</body>
```

```
> $("#content >div").is(".test")
< true
> $("#content >div").is("#a")
< true
> $("#content >div").is("#hello")
< false
```

# 필터링 메서드(10)



- `map()` : 선택된 요소에 대해 인자로 전달한 함수를 매번 실행하여 리턴한 값을 모아서 담은 jQuery 객체를 리턴

```
<body>
  <div id="content">
    <div id="a">김수한무</div>
    <div>거북이와 두루미</div>
    <div class="test">삼천갑자 동방삭</div>
    <div class="test main">치치카포 사리사리센타</div>
    <div>위리위리 세브리깁</div>
    <div>무두셀라 구름이</div>
    <div id="b">허리케인에 담벼락</div>
    <div class="main">담벼락에 서생원</div>
    <div class="test">서생원에 고양이</div>
    <div id="a">고양이에 바둑이</div>
    <div>바둑이는 돌돌이</div>
  </div>
```

[ 예제 08-06 ]

```
01: <script type="text/javascript">
02: $(document).ready(function() {
03:   var jQueryObject = $("#content > div").map(function(i, ele) {
04:     return ele.innerHTML;
05:   });
06:   var arr = jQueryObject.get();
07:   console.log(arr);
08: });
09: </script>
```



```
▼ Array[11] ⓘ
  0: "김수한무"
  1: "거북이와 두루미"
  2: "삼천갑자 동방삭"
  3: "치치카포 사리사리센타"
  4: "위리위리 세브리깁"
  5: "무두셀라 구름이"
  6: "허리케인에 담벼락"
  7: "담벼락에 서생원"
  8: "서생원에 고양이"
  9: "고양이에 바둑이"
 10: "바둑이는 돌돌이"
  length: 11
  ▶ proto : Array[0]
```

# 필터링 메서드(11)



## ■ map() 메서드의 더 현실적인 예제(예제 08-07)

```
20: <h2>지금 먹고 싶은 과일은?</h2>
21: <div id="content">
22:   <input type="checkbox" class="fruit" value="1">사과,
23:   <input type="checkbox" class="fruit" value="2">배,
24:   <input type="checkbox" class="fruit" value="3">키위,
25:   <input type="checkbox" class="fruit" value="4">포도<br/>
26:   <input type="checkbox" class="fruit" value="5">딸기,
27:   <input type="checkbox" class="fruit" value="6">귤,
28:   <input type="checkbox" class="fruit" value="7">수박<br/>
29: </div>
30: <button id="save">선택 저장</button>
31: <div id="list"></div>
```

### 지금 먹고 싶은 과일은?

☒ 사과, ☐ 배, ☐ 키위, ☐ 포도

☐ 딸기, ☒ 귤, ☒ 수박

선택 저장

1, 6, 7

```
09: $("#save").click(function() {
10:   var arr = $("input.fruit:checked").map(function(i, element) {
11:     return element.value;
12:   }).get();
13:
14:   $("#list").html(arr.join(", "));
15: });
```

# 트리 횡단 탐색 메서드(1)



- 현재 선택된 요소 노드 중심으로 상대 경로로 다른 노드를 선택하는 기능을 제공한다.

[ 표 08-02 : 트리 횡단 탐색 메서드 ]

메서드	설명
find( )	현재 선택된 요소의 후손 중에서 지정된 선택자로 선택될 수 있는 요소를 모두 선택한다.
prev( )	현재 선택된 요소의 이전 요소를 선택한다(형제 요소 중 이전 요소). 선택자를 지정할 경우 필터링할 수 있다.
prevAll( )	현재 선택된 요소의 형제 요소 중 이전 요소 모두 선택한다.
prevUntil( )	현재 선택된 요소의 형제 요소 중 선택된 요소에서 시작하여 이전으로 거슬러 올라가면서 지정된 선택자에 부합되는 요소를 만날 때까지의 요소를 모두 선택한다. 추가로 필터를 지정할 수 있다.
next( )	현재 선택된 요소의 다음 요소를 선택한다(형제 요소 중 다음 요소). 선택자를 지정할 경우 필터링할 수 있다.
nextAll( )	현재 선택된 요소의 형제 요소 중 다음 요소 모두 선택한다.
nextUntil( )	prevUntil과 동일한 기능이다. prevUntil처럼 거슬러 올라가면서 선택하는 것이 아니라 다음으로 이동하면서 선택한다는 차이점이 있다.
siblings( )	현재 선택된 요소의 형제 요소를 선택한다.
parent( )	현재 선택된 요소의 부모 요소를 선택한다.
parents( )	현재 선택된 요소의 조상 요소를 모두 선택한다. 최상위 <html> 요소까지 모두 선택한다. 선택자를 지정하여 필터링할 수 있다.
closest( )	parents( )와 유사하지만 조상 요소 중 지정된 선택자에 부합되는 요소 하나만을 선택한다. 조상 중 현재 선택된 요소와 가장 가까운 관계에 있는 것 하나만을 선택한다는 것이 parents( )와의 차이이다.
children( )	현재 선택된 요소의 모든 직계 자식 요소를 선택한다. 선택자로 필터링할 수 있다.

# 트리 횡단 탐색 메서드(2)



## ■ find() 메서드

- 자주 사용하는 메서드. 반드시 알아두자.
- 기존 선택된 노드의 후손들 중에서 찾아 선택함.

```
$("#a").css("border", "solid 1px black").find("div").css("background-color", "orange");
```

## ■ 예제 08-08

- 선택된 jQuery 객체를 캐싱하고 find를 여러번 호출
  - 문서 전체에서 찾아 선택하는 횟수 최소화

```
$("#foods tr:odd").css("background-color", "aqua");  
$("#foods tr:even").css("background-color", "#FFCCCC");  
$("#foods tr:first").css("background-color", "purple").css("color", "yellow");  
$("#foods tr:eq(3)").css("font-style", "italic");
```

개선



```
var foods = $("#foods");  
foods.find("tr:odd").css("background-color", "aqua");  
foods.find("tr:even").css("background-color", "#FFCCCC");  
foods.find("tr:first").css("background-color", "purple").css("color", "yellow");  
foods.find("tr:eq(3)").css("font-style", "italic");
```

## 트리 횡단 탐색 메서드(3)



### ■ find()의 또다른 사용 형태

- "id가 a 또는 c의 후손 중에서 div 태그를 모두 선택한다"
  - 아마도 이렇게? `$("#a,#c div")`
- 선택자 우선순위가 후손 선택자가 또 높기 때문에 다음과 같이 작동함
  - "id가 a인 요소를 선택하거나 id가 c인 요소의 후손 중에서 div 태그를 선택한다."
- 처음 기대했던 대로 작동하려면 find() 메서드를 사용할 수 있음
  - `$("#a,#c").find("div")`
- find() 메서드를 사용한 쪽이 훨씬 명확해보임

```
12: $("#a, #c").css("border", "solid 1px black").find("div").css("background-color", "orange");
13: //다음 코드는 $("#a").add("#c div")와 동일함.
14: $("#a, #c div").css("text-decoration", "underline");
```



## 트리 횡단 탐색 메서드(4)



- find() 메서드 대신 contex를 이용할 수 있다. 아래 코드는 동일한 결과이다.

```
var content = $("#content");  
content.find(".test").css("color", "blue");
```

```
var content = $("#content");  
$(".test", content).css("color", "blue");
```

- jQuery 함수 형태

```
jQuery = function( selector, context ) {  
    // The jQuery object is actually just the init constructor 'enhanced'  
    // Need init if jQuery is called (just allow error to be thrown if not included)  
    return new jQuery.fn.init( selector, context );  
}
```



# 트리 횡단 탐색 메서드(5)



## ■ prev(), next(),siblings() 메서드

- 현재 선택된 요소를 중심으로 형제 요소 중에서 이전 요소, 다음 요소를 선택하는 메서드

```
20: <div id="content">
21:   <div id="a">김수한무</div>
22:   <div>거북이와 두루미</div>
23:   <div class="test">삼천갑자 동방삭</div>
24:   <div class="test main">치치카포 사리사리센타</div>
25:   <div>워리워리 세브리깡</div>
26:   <div>무두셀라 구름이</div>
27:   <div id="b">허리케인에 담벼락</div>
28:   <div class="main">담벼락에 서생원</div>
29:   <div class="test">서생원에 고양이</div>
30:   <div id="a">고양이에 바둑이</div>
31:   <div>바둑이는 둘둘이</div>
32: </div>
```

```
10: <script type="text/javascript">
11: var current;
12: $(document).ready(function() {
13:   current = $("#b").css("background-color", "aqua");
14:   current.prev().css("text-decoration", "underline");
15:   current.next().css("text-decoration", "line-through");
16: });
17: </script>
```



김수한무  
거북이와 두루미  
삼천갑자 동방삭  
치치카포 사리사리센타  
워리워리 세브리깡  
무두셀라 구름이  
허리케인에 담벼락  
~~담벼락에 서생원~~  
~~서생원에 고양이~~  
~~고양이에 바둑이~~  
~~바둑이는 둘둘이~~

# 트리 횡단 탐색 메서드(6)



## ■ `current.prev("p")`

아무것도 선택되지 않는다. `prev()` 메서드의 파라미터로 선택자를 전달하면, 이전의 요소가 선택자에 부합되지 않으면 요소를 선택하지 않는다.

## ■ `current.prevAll()`

현재 요소와 형제 관계에 있는 모든 이전 요소를 선택한다. 메서드의 파라미터로 선택자를 전달하면 선택자에 부합되는 요소로 필터링한다.

## ■ `current.prevUntil("#a")`

현재 요소의 `prev()`부터 거슬러 올라가면서 선택자 "#a"에 부합되는 요소 직전까지만 선택한다.

## ■ `current.prevUntil("#a", ".test")`

`prevUntil()` 메서드의 두 번째 파라미터는 필터링하기 위한 것이다. 현재 요소에서 첫 번째 파라미터로 전달된 선택자에 부합되는 것을 요소의 범위 안에서 다시 필터링한 결과를 리턴한다.

김수한무 → #a

거북이와 두루미

삼천갑자 동방삭

치치카포 사리사리센타

워리워리 세브리깁

무두셀라 구름이

허리케인에 담벼락

담벼락에 서생원

서생원에 고양이

고양이에 바둑이

바둑이는 돌들이

current.prevUntil("#a")

current

`current.prevUntil("#a", ".test")`은 `prevUntil("#a")`의 요소 중 `.test` 선택자를 만족하는 것만으로 필터링한다.

- `next()`, `nextAll()`, `nextUntil()` 메서드는 `prev()`, `prevAll()`, `prevUntil()` 메서드와 동일한 관점

# 트리 횡단 탐색 메서드(7)



## ■ parent(), parents(), closest() 메서드

- 현재 선택된 요소를 중심으로 부모 및 조상 요소를 선택하는 메서드
  - parent() : 직계 부모 요소 선택
  - parents() : 현재 요소의 모든 조상 요소 선택
  - closests() : 현재 요소의 조상 요소 중 파라미터로 전달된 선택자에 부합되는 조상 요소 하나만 선택

## 예제 08-11

```
07: <script type="text/javascript">
08: var current;
09: $(document).ready(function() {
10:   current = $("li.item-1").css("background-color", "aqua");
11: });
12: </script>
13: </head>
14: <body>
15:   <ul class="level-1">
16:     <li class="item-i">I</li>
17:     <li class="item-ii">II
18:       <ul class="level-2">
19:         <li class="item-a">A</li>
20:         <li class="item-b">B
21:           <ul class="level-3">
22:             <li class="item-1">1</li>
23:             <li class="item-2">2</li>
24:             <li class="item-3">3</li>
25:           </ul>
26:         </li>
27:         <li class="item-c">C</li>
28:       </ul>
29:     </li>
30:     <li class="item-iii">III</li>
31:   </ul>
32: </body>
```

# 트리 횡단 탐색 메서드(8)



## 예제 08-11 대상으로 브라우저 콘솔에서 실행

- `current.parent( )`  
직계 부모인 21행의 `<ul>` 요소가 선택된다. 만일 파라미터로 선택자를 전달하는 경우 선택자에 부합되지 않으면 아무런 요소도 선택하지 않는다.
- `current.parents( )`  
모든 조상 요소를 상위 요소로 거슬러 올라가면서 찾아 모두 선택한다.
- `current.parents("li")`  
모든 조상 요소 중 지정한 선택자에 부합되는 요소를 모두 찾아 선택한다.
- `current.closest("li")`  
현재 요소의 조상 요소 중에서 지정한 선택자를 만족하는 가장 가까운 요소 하나를 선택한다.

- I
- II
  - A
  - B
    - 1
    - 2
    - 3
  - C
- III

Elements Console Sources Network Timeline Profiles Resources Security Audits

<top frame> ☐ Preserve log

```
> current.parent()
< [ > <ul class="level-3">...</ul> ]

> current.parents()
< [ > <ul class="level-3">...</ul>, > <li class="item-b">...</li>, > <ul class="level-2">...</ul>,
    > <li class="item-ii">...</li>, > <ul class="level-1">...</ul>, > <body>...</body>, <html lang=
    > <head>...</head>
    > <body>...</body>
    </html>

> current.parents("li")
< [ > <li class="item-b">...</li>, > <li class="item-ii">...</li> ]

> current.closest("li")
< [ <li class="item-1" style="background-color: aqua;">1</li> ]
```

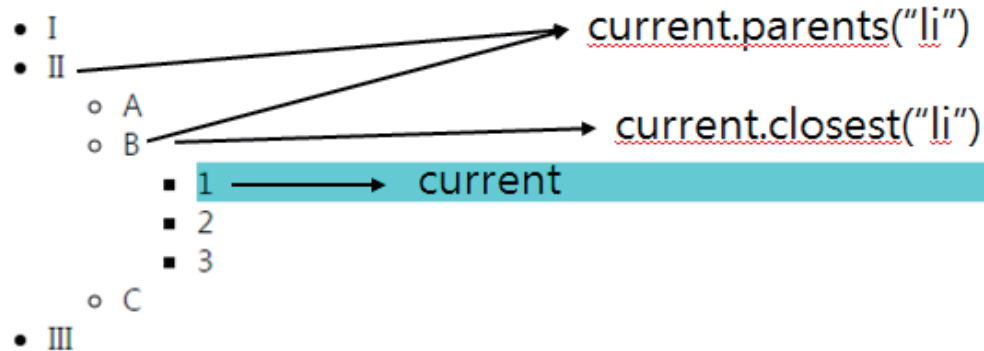
# 트리 횡단 탐색 메서드(9)



- parents()와 closest()의 차이점
  - 아래 두 코드는 같은 결과임

```
current.parents("li").first()
```

```
current.closest("li")
```



# 트리 횡단 탐색 메서드(10)



## ■ children() 메서드

- 현재 선택된 요소의 직계 자식 요소를 선택
  - 자식 노드들중 요소노드만 선택함. 19행의 '김수한무'는 텍스트노드이므로 선택(x)

```
10: <script type="text/javascript">
11: $(document).ready(function() {
12:   $("#a").children().css("text-decoration", "underline");
13:   $("#a").children(".test").css("font-style", "italic");
14: });
15: </script>
16: </head>
17: <body>
18:   <div id="a">
19:     김수한무
20:     <div>거북이와 두루미</div>
21:     <div class="test">삼천갑자 동방삭</div>
22:   </div>
23:   <div id="b">
24:     <div>치치카프 사리사리센타</div>
25:     <div>워리워리 세브리깡</div>
26:     <div>무두셀라 구름이</div>
27:   </div>
28: </body>
29: </html>
```

김수한무  
거북이와 두루미  
삼천갑자 동방삭  
치치카프 사리사리센타  
워리워리 세브리깡  
무두셀라 구름이

# 기타 횡단 탐색 메서드(1)



## ■ 5개 메서드

[ 표 08-03 : 기타 횡단 탐색 메서드 ]

메서드	설명
add( )	현재 선택된 요소 집합과 새로이 선택한 요소의 합집합을 만든다.
addBack( )	현재 선택된 요소 집합과 이전에 선택된 요소 집합을 합하여 새로운 집합을 만든다. 이때 이전에 선택된 집합의 요소 중 선택자로 필터링할 수 있다.
end( )	직전 단계에서 선택되었던 요소를 선택한다. jQuery 객체의 prevObject 속성을 이용한다.
contents( )	직계 자식 중에서 콘텐츠를 모두 선택한다. 요소뿐만 아니라 주석, 텍스트도 콘텐츠로 간주되므로 주의한다. 이 메서드는 전달할 수 있는 파라미터가 없다.
each( )	jQuery 객체가 선택하고 있는 객체를 이용해 반복 처리 작업을 수행한다. 함수를 파라미터로 전달하는데 이 함수는 인덱스 번호, 아이템을 인자로 전달받는다.

## 기타 횡단 탐색 메서드(2)



### ■ add(), addBack(), end() 메서드

- 집합의 개념을 적용하면 쉽게 이해할 수 있음
- jQuery 객체의 prevObject 속성을 이해하고 있다면 쉽게 알 수 있다.

```
16: <body>
17:   <div id="content">
18:     <div id="a">김수한무</div>
19:     <div>거북이와 두루미</div>
20:     <div class="test">삼천갑자 동방삭</div>
21:     <div class="test main">치치카포 사리사리센타</div>
22:     <div>우리우리 세브리깁</div>
23:     <div>무두셀라 구름이</div>
24:     <div id="b">허리케인에 담벼락</div>
25:     <div class="main">담벼락에 서생원</div>
26:     <div class="test">서생원에 고양이</div>
27:     <div id="a">고양이에 바둑이</div>
28:     <div>바둑이는 돌들이</div>
29:   </div>
30: </body>
```

```
09: var current = $("#content > div").eq(4).next();
10: current.addBack().css("font-weight", "bold");
11: current.add(".main").css("background-color", "yellow");
12: current.end().css("font-style", "italic");
```

김수한무  
거북이와 두루미  
삼천갑자 동방삭  
치치카포 사리사리센타  
우리우리 세브리깁  
무두셀라 구름이  
허리케인에 담벼락  
담벼락에 서생원  
서생원에 고양이  
고양이에 바둑이  
바둑이는 돌들이

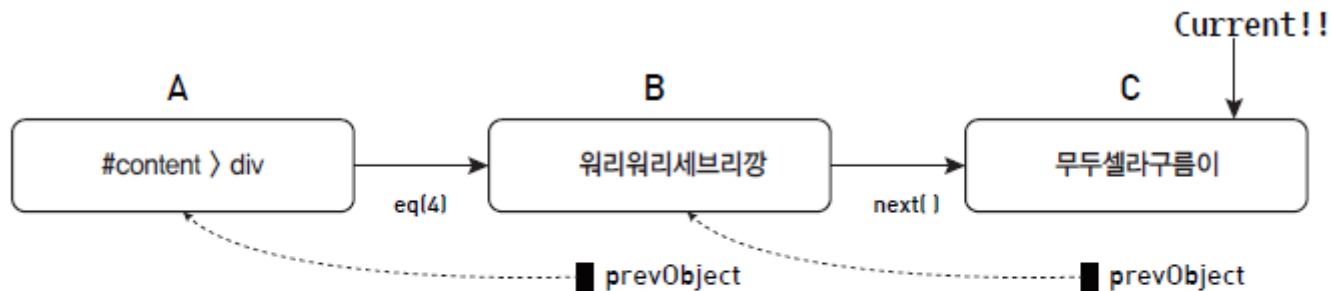


# 기타 횡단 탐색 메서드(3)



## ■ prevObject 속성 확인예

```
> var a = $("#content > div").eq(4).next();  
< undefined  
> a  
< [ <div style="font-weight: bold; background-color: yellow;">무두셀라 구름이</div>]  
> a.prevObject  
< [ <div style="font-weight: bold; font-style: italic;">워리워리 세브리깡</div>]
```



```
current.addBack() : B U C  
current.add(".main") : C U $(".main")  
current.end() : B
```

# 기타 횡단 탐색 메서드(4)



## ■ contents() 메서드

- 현재 선택된 노드의 직계 자식 노드 모두 선택
  - children() 메서드는 직계 자식 요소 노드만 선택하는 반면 contents() 메서드는 텍스트 노드, 주석 노드까지 선택
- 필터링할 수 있는 파라미터를 사용할 수 없으므로 추가로 filter() 메서드를 이용해 걸러내는 경우가 많다.

```
21: <button id="toBold">텍스트 노드를 찾아 볼드체로</button>
22: <hr />
23: <div id="content">
24:     김수한무
25:     <div>거북이와 두루미</div>
26:     삼천갑자 동방삭
27:     <div>치치카포 사리사리센타</div>
28:     워리워리 세브리깁
29: </div>
```

```
11: $(document).ready(function() {
12:     $("#toBold").click(function() {
13:         $( "#content" ).contents().filter(function(){
14:             return this.nodeType === 3;
15:         }).wrap( "<b></b>" );
16:     });
17: });
```

텍스트 노드를 찾아 볼드체로

김수한무  
거북이와 두루미  
삼천갑자 동방삭  
치치카포 사리사리센타  
워리워리 세브리깁

# 기타 횡단 탐색 메서드(5)



## ■ each() 메서드

- jQuery 객체가 선택하고 있는 요소를 이용해 반복문을 실행하면서 인자로 전달된 함수를 호출
- for 문 대신에 사용 가능

```
16: <div id="content">
17:   <div id="a">김수한무</div>
18:   <div>거북이와 두루미</div>
19:   <div class="test">삼천갑자 동방삭</div>
20: </div>
```

0--->김수한무

1--->거북이와 두루미

2--->삼천갑자 동방삭

```
08: $(document).ready(function() {
09:   $("#content > div").each(function(i, item) {
10:     console.log(i + "--->" + item.innerHTML);
11:   });
12: });
```

# 기타 횡단 탐색 메서드(6)



- each() 의 성능
  - for 문 보다 좋지 않다.
- forEach()
  - 배열에서만 사용 가능. 유사 배열은 지원하지 않음
  - 전달하는 함수의 인자 순서가 \$.each()와 뒤바뀌어 있음

```
var arr = [  
    { name: "안재홍", score : 192400 },  
    { name: "조진웅", score : 200120 },  
    { name: "박보검", score : 120340 },  
    { name: "이제훈", score : 151260 }  
];  
console.log("-----");  
$.each(arr, function(i, item) {  
    console.log(item.name + " : " + item.score);  
});
```

# 기타 횡단 탐색 메서드(7)



- \$.each() 가 for보다 성능이 좋지 않음에도 사용하는 이유?
  - 단순히 반복문을 대체하는 것은 아님
  - 반복문에서의 클로저로 인한 문제 해결을 위한 것임
  - 예제 03-27 소환!!

[ 예제 03-16 : 예제 03-27 소환 ]

```
01: <!DOCTYPE html>
02: <html lang="">
03: <head>
04: <meta charset="utf-8">
05: <title>03-27 리뷰</title>
06: <script type="text/javascript">
07: window.onload = function() {
08:     var list = document.getElementsByTagName("input");
09:     for (var i=0; i < list.length; i++) {
10:         list[i].onclick = function() {
11:             alert((i+1) + "번째 버튼 클릭!");
12:         }
13:     }
14: }
15: </script>
16: </head>
```

```
17: <body>
18:     <input id="b1" class="test" type="button" value="1번째 버튼" />
19:     <input id="b2" class="test" type="button" value="2번째 버튼" />
20:     <input id="b3" class="test" type="button" value="3번째 버튼" />
21: </body>
22: </html>
```

# 기타 횡단 탐색 메서드(8)




## ■ 3장에서 의 해결 방법

```
<script type="text/javascript">
window.onload = function() {
    var list = document.getElementsByTagName("input");
    for (var i=0; i < list.length; i++) {
        setIndex(i);
    }

    function setIndex(k) {
        list[k].onclick = function() {
            alert((k+1) + "번째 버튼 클릭!");
        }
    }
}
</script>
```

## ■ \$.each() 메서드로 해결

```
07: <script type="text/javascript">
08: $(document).ready(function() {
09:     var btns = $("input.test");
10:     for (var i=0; i < btns.length; i++) {
11:         $(btns[i]).click(function() {
12:             alert((i+1) + "번째 버튼 클릭!!!");
13:         });
14:     }
15: });
16: </script>
```



```
01: <script type="text/javascript">
02: $(document).ready(function() {
03:     var btns = $("input.test");
04:     btns.each(function(i, item) {
05:         $(item).click(function() {
06:             alert((i+1) + "번째 버튼 클릭!!!");
07:         });
08:     });
09: });
10: </script>
```

# 정리



## ■ 선택 성능 향상을 위해서 횡단 탐색 메서드를 사용하자

- 문서 전체에서 찾아 선택하는 작업의 횟수를 줄인다.
- 성능 차이가 꽤 크다. 특히 find() 메서드

## ■ 명확한 선택

- 선택자만으로 표현하기 힘든 것도 횡단 탐색메서드를 이용하면 깔끔하게 선택!!
  - 특히add(), find() 메서드!!
  - 복잡한 선택자를 한번에 작성하지 말고 기본 선택자로 선택한 후 횡단 탐색 메서드로 마무리한다.