선택자와 성능

- jQuery는 성능이 좋지 않다?
  - Pure Javascript로 개발했을 때보다 좋을리는 없지만...
  - 조금만 신경쓰면 꽤 괜찮은 성능을 낼 수 있음
- 성능에 큰 영향을 주는 부분
  - 선택자!! 횡단 탐색!!
    - 선택자와 횡단 탐색 메서드를 적절히 조합하여 사용하면 성능 향상에 도움이 됨

횡단 탐색 메서드 실행 속도

이 장은 선택자나 횡단 탐색 메서드의 실행 속도를 알아보는 목적으로 작성했다. 예제를 직접 작성해서 실행해 보면 좋지만 여건이 여의치 않다면, 제공하는 샘플 예제를 살펴보고 실행해보기만 해도 된다.

기본 선택자의 속도(1)

실행 속도를 알아보기 위한 기본 템플릿!

```
07: <script type="text/javascript">
08: $(document).ready(function() {
09:   console.time("create bulk element");
10:   var str = "";
11:   for (var i=0; i < 100; i++) {
12:      var a = "a"+i;
13:      str += "<div id='"+a+"'>";
14:      for (var j=0; j < 100; j++) {
15:         var b = "b_" + i + "_" +j;
16:         str +="<div id='" + b + "'";
17:         if (j%4==0) {
18:            str+= " class='main'";
19:         }
20:         str+=">" + i + "," + j + "</div>";
21:      }
22:      str+= "</div>"
23:   }
24:   $("#content").html(str);
25:   $("#b_30_50").addClass("test").wrap("<span></span>");
26:   console.timeEnd("create bulk element");
27:
28: });
29: </script>
30: </head>
31: <body>
32:     <div id="content"></div>
```

```
<div id="a0">
   <div id="b_0_0" class="main">0,0</div>
   <div id="b_0_1">0,1</div>
   <div id="b_0_2">0,2</div>
   <div id="b_0_3">0,3</div>
   <div id="b_0_4" class="main">0,4</div>
   <div id="b_0_5">0,5</div>
   <div id="b_0_6">0,6</div>
   <div id="b_0_7">0,7</div>
   <div id="b_0_8" class="main">0,8</div>
   ......(100개의 div)
</div>
```

- 위와 같은 요소를 100개 생성하여 div#content에 추가함

기본 선택자의 속도(2)

예제 09-02

- 기본 선택자들의 실행 속도 측정

[예제09-02]

```
01:     //---id로 1000번 찾기
02:   console.time("find by id 1000");
03:   for (var i=0; i < 1000; i++) {
04:      $("#b_30_50").css("color", "blue");
05:   }
06:   console.timeEnd("find by id 1000");
07:
08:   //--class name으로 1000번 찾기
09:   console.time("find by class 1000");
10:   for (var i=0; i < 1000; i++) {
11:      $(".test").css("color", "blue");
12:   }
13:   console.timeEnd("find by class 1000");
14:
15:   //--span 태그로 1000번 찾기
16:   console.time("find by tagname 1000");
17:   for (var i=0; i < 1000; i++) {
18:      $("span").css("color", "blue");
19:   }
20:   console.timeEnd("find by tagname 1000");
```

```
create bulk element: 67.786ms
find by id 1000: 21.843ms
find by class 1000: 31.507ms
find by tagname 1000: 33.015ms
```

- 브라우저마다 실행시간의 차이는 있겠지만 대체로 다음과 같은 결과

ID 선택자 >= 클래스 선택자 = 태그 이름 선택자

- 이 3가지 선택자가 가장 빠르다!!
- 가능하다면 이 3가지 선택자는 단독으로 사용하는 것이 바람직하다.

기본 선택자의 속도(3)

- 최종적으로 선택된 요소가 같아도 실행 속도의 차이가 날 수 있다.

[예제 09-03]

```
01:      //---id로 1000번 찾기
02:   console.time("find by id 1000");
03:   for (var i=0; i < 1000; i++) {
04:      $("#b_50_50").css("color", "blue");
05:   }
06:   console.timeEnd("find by id 1000");
07:
08:
09:   //--- 특성 선택자(Attribute Selector)
10:   console.time("find by attribute 1000");
11:   for (var i=0; i < 1000; i++) {
12:      $("[id='b_50_50']").css("color", "blue");
13:   }
14:   console.timeEnd("find by attribute 1000");
```

```
create bulk element: 77.745ms
find by id 1000: 19.998ms
find by attribute 1000: 782.589ms
```

후손 선택자와 find() 메서드

- 둘 중의 어느 쪽이 더 좋은 성능
  - A : $("#a50 .main")
  - B : $("#a50").find(".main")
  - B가 월등히 좋은 성능을 낸다.

[예제 09-04]

```
01:     //---후손 선택자
02:  console.time("select by descendant selector 1000");
03:  for (var i=0; i < 1000; i++) {
04:     $("#a50  .main").css("color", "blue");
05:  }
06:  console.timeEnd("select by descendant selector 1000");
07:
08:  //---find() 메서드
09:  console.time("find method 1000");
10:  for (var i=0; i < 1000; i++) {
11:     $("#a50").find(".main").css("color", "blue");
12:  }
13:  console.timeEnd("find method 1000");
14:
15:  //---객체 cache 후 find() 메서드
16:  console.time("cache and find method 1000");
17:  var a50 = $("#a50");
18:  for (var i=0; i < 1000; i++) {
19:     a50.find(".main").css("color", "blue");
20:  }
21:  console.timeEnd("cache and find method 1000");
```

```
create bulk element: 76.154ms
select by descendant selector 1000: 725.268ms
find method 1000: 91.387ms
cache and find method 1000: 90.362ms
```

- 특정 요소 내에서 반복해서 선택할 경우 후손 선택자를 반복해서 사용하지 말고, 선택 범위를 미리 선택하여 캐싱하고 find() 메서드를 활용하는 편이 바람직하다.

jQuery 확장 선택자(1)

선택자란?

- HTML 문서에 적용할 디자인을 CSS로 정의하고 이 디자인을 지정한 요소에 적용하는 방법을 위해 사용되었다.
- 이 선택자를 jQuery에서 요소 선택을 위해 사용
- 하지만 모든 것이 CSS 표준 스펙을 만족하지는 않음

jQuery 확장 선택자

- CSS 표준 스펙에는 없지만 jQuery에서 사용하는 선택자를 jQuery 확장 선택자(jQuery Extension selector)라고 부름

:animated	:eq( )	:even
:odd	:first	:last
:gt( )	:lt( )	:header
:not( )	:has( )	:parent
:button	:checkbox	:file
:image	:input	:password
:radio	:reset	:selected
:submit	:text	:hidden
:visible		

jQuery 확장 선택자(2)

- (이어서)

jQuery 선택자

jQuery에서 사용하는 선택자를 사용했을 때 jQuery 내부에서는 가능한 브라우저가 제공하는 네이티브 DOM 메서드를 이용하려고 한다. 그것이 더 성능이 좋기 때문이다. ID 선택자를 사용하는 경우 getElementById( ) 메서드를 사용하며, 클래스 선택자인 경우는 getElementsByClassName( ) 메서드를 이용한다. ID, 클래스, 태그 이름 선택자와 같은 기본 선택자가 아니라면 querySelector( ) 또는 querySelectorAll( ) 메서드가 사용되지만 CSS 표준 스펙의 선택자가 아닌 경우는 요소 하나하나를 찾아나가는 순회 탐색이 실행되기 때문에 매우 느리다.

- 한마디로 jQuery 확장 선택자는 가능하다면 사용하지 않는 것이 바람직하다.
  - 불가피하게 사용해야만 한다면 기본 선택자로 찾는 범위를 좁혀 놓고 횡단 탐색 메서드를 사용해 찾는다.

jQuery 확장 선택자(3)

예제 09-05

```
25:     //---- 입력폼 필터로 찾기
26:     console.time("text form filter 1000");
27:     for (var i=0; i < 1000; i++) {
28:         $(":text").css("color", "blue");
29:     }
30:     console.timeEnd("text form filter 1000");
31:
32:     //----태그명 + 입력폼 필터
33:     console.time("tag name & text form filter 1000");
34:     for (var i=0; i < 1000; i++) {
35:         $("input:text").css("color", "blue");
36:     }
37:     console.timeEnd("tag name & text form filter 1000");
39:     //----태그명 -> filter 메서드
40:     console.time("tag name -> filter method 1000");
41:     for (var i=0; i < 1000; i++) {
42:         $("input").filter(":text").css("color", "blue");
43:     }
44:     console.timeEnd("tag name -> filter method 1000");
45:
46:     //---- 클래스 선택자로 찾기
47:     console.time("class selector 1000");
48:     for (var i=0; i < 1000; i++) {
49:         $(".input_name").css("color", "blue");
50:     }
51:     console.timeEnd("class selector 1000");
```

```
create bulk element: 72.249ms
text form filter 1000: 6960.439ms
tag name & text form filter 1000: 955.032ms
tag name -> filter method 1000: 431.690ms
class selector 1000: 308.922ms
```

- jQuery 확장 선택자인 text form filter 가 가장 나쁜 성능!!
- 기본 선택자와 결합시키거나 filter 메서드를 사용했을 때 성능이 개선된 것을 볼 수 있음
- 역시 가능하자면 기본 선택자를 단독으로 사용하는 것이 좋다.

- 아쉽지만 :selected 필터도 jQuery 확장 선택자이다.

정리(1)

- 메서드 체이닝 또는 jQuery 캐시 객체를 사용하자.

```
$("#content").css("color", "brown");
$("#content").css("background-color", "aqua");
$("#content").css("font-weight", "bold");
```

```
$("#content")
    .css("color", "brown")
    .css("background-color", "aqua")
    .css("font-weight", "bold");
```

```
var c = $("#content");
c.css("color", "brown");
c.css("background-color", "aqua");
c.css("font-weight", "bold");
```

```
var option = { "color":"brown", "background-color":"aqua", "font-weight":"bold" };
$("#content").css(option);
```

정리(2)

선택자를 반드시 사용해야 하는가?

- 반드시 선택자를 사용할 필요는 없다.
  - 이미 선택된 요소가 있다면 굳이 사용할 필요가 없다.

```
$("#result").html($(this).attr("data-no"));
 --> $("#result").html(this.attributes["data-no"].value);
```

- 성능보다 개발 생산성이 더 중요하다면 jQuery 사용을 더욱 권장함
  - 크로스 브라우저 기능 지원

정리(3)

- 성능을 최우선으로 고려해야 한다면 jQuery 선택자와 함수를 사용하는 대신 네이티브 API를 사용할 것을 권장
  - document 객체의 getElementById(), getElementsByTagName(), querySelector(), querySelectorAll() 등의 메서드가 훨씬 성능이 좋기 때문이다

Test runner

Done. Ready to run again. Run again

Testing in Chrome 48.0.2564.116 32-bit on Windows NT 6.3 64-bit		
Test		**Ops/sec**
jQuery ID Selector	`$("#testid");`	1,209,191 ±1.27% 94% slower
getElementById	`document.getElementById("testid");`	18,980,357 ±0.83% fastest
jQuery class Selector	`$(".testclass");`	319,121 ±1.28% 98% slower
Native querySelector	`document.querySelector(".testclass");`	1,474,017 ±0.64% 92% slower
Create jQuery from Native	`$(document.getElementById("testid"));`	1,871,118 ±5.25% 90% slower
Native querySelector to jQuery Object	`$(document.querySelector(".testclass"));`	558,564 ±0.00% 97% slower

정리(4)

- 다시 한번 정리
  - 선택자의 실행 속도는 제각각이므로 가능하다면 빠른 선택자를 사용하자. 일반적으로 ID 선택자가 가장 빠르다.
  - 느린 선택자를 사용할 때는 찾을 범위를 좁히는 것이 바람직하다. 특히 후손 선택자나 AND 조합, 횡단 탐색 메서드를 고려한다.
  - jQuery 확장 선택자는 실행 속도가 매우 느리다. 따라서 표준 선택자로 찾은 후에 횡단 탐색(Traverse) 메서드를 사용하는 것이 가장 효과적이다.
  - 이미 선택된 요소가 있다면 $() 함수를 호출할 필요 없이 요소에 직접 접근하는 것이 성능을 높이는 방법이다.
  - 성능이 최우선 고려 사항이라면 네이티브 API를 사용하자. jQuery를 사용하는 것보다 빠르다. 다만 성능이 중요한 고려 사항이 아니라면 jQuery를 사용하는 것이 개발 생산성 향상, 크로스 브라우저 지원 등의 장점을 얻을 수 있다.