jQuery 의 기능적인 특징을 핵심 키워드만 뽑아서 정리하자면, 막강한 CSS 셀렉터, 크로스 브라우저 지원, 메서드 체인, Ajax 지원, 풍부한 플러그 인 지원

1. jQuery 셀렉터 : 기본

jQuery 의 가장 강력한 부분은 HTML DOM 을 마음대로 트래버스 즉, 순회 탐색할 수 있다는 것인데요. 놀랍게도, CSS 셀렉터를 사용하여 원하는 개체를 탐색할 수 있습니다. 다음과 같이 css 파일에서 사용했던 표현식이 바로 CSS 셀렉터이니까요.

```
div p {
    font-color:red;
}

#loginID {
    font-weight:bold;
    background:yellow;
}

.Columns {
    padding:10px;
    background:white;
}
```

div p 라는 셀렉터는 현재 문서 상에서 div 요소의 자식으로 존재하는 모든 p 요소들에 적용되며, #loginID 는 loginID 라는 id 값을 가진 요소에, .Columns 는 class 어트리뷰트 값으로 Columns 를 갖는 모든 요소에 적용되는 것이죠.
위의 3 가지 셀렉터를 다음과 같이 사용할 수 있습니다.

$("div p") 혹은 jQuery("div p")
$("#loginID") 혹은 jQuery("#loginID")
$(".Columns") 혹은 jQuery(".Columns")

그리고, 각각의 표현식은 각 DOM 요소의 확장 개체인 jQuery 개체 집합을 반환합니다. DOM 요소를 직접 반환해주는 것이 아니라 그의 래퍼(Wrapper)인 jQuery 개체로 반환해 주기 때문에 직접 DOM 요소를 제어할 때보다 훨씬 편하고 쉽게 개체를 제어할 수 있다는 장점이 있습니다. 예를 들어, $("div p").hide() 와 같은 명령을 사용하면, 현재 문서 상에서 div 요소의 자식으로 존재하는 모든 p 태그집합들은 눈에서 보이지 않게 됩니다. hide()라는 명령은 jQuery 개체가 지원하는 명령이며, 추후 알아보게 될 것인데요. jQuery 셀렉터에 의해 반환되는 개체가 일반 DOM 개체라면 이러한 명령을 사용할 수 없겠지만, jQuery 개체 집합으로 반환되기에 그러한 명령을 사용할 수 있게 되는 것입니다.
모든 요소를 선택하기 위해서는 $("*")와 같은 표현을 사용할 수 있습니다.
또한, jQuery 는 이러한 기본적인 CSS 셀렉터 외에 고급 CSS 셀렉터도 지원합니다. 예를 들면, 계층 셀렉터, 일반 필터 셀렉터, 어트리뷰트 필터 셀렉터 등이 바로 그것인데요. 다음은 계층 셀렉터의 예입니다.

p >a : p 요소 바로 아래 자식인 a 요소(하이퍼링크)와 일치된다.
div + p : div 요소의 바로 다음에 나오는 형제 p 요소와 일치된다.
div ~ p : div 요소의 다음에 나오는 모든 형제 p 요소와 일치된다.

다음과 같은 html 이 존재한다면,

```
<p>
    <span>
        <a href="1.aspx">1.aspx</a>
    </span>
    <br />
    <a href="2.aspx">2.aspx</a>
</p>
```

$("p a")는 1.aspx 링크와 2.aspx 링크 모두를 선택하지만, $("p > a")는 2.aspx 링크만을 선택한다는 것이죠.
div + p 및 div ~ p 와 같은 경우는 자식이 아니라 형제 요소와 연관이 있습니다.

```
<div>앨범 목록입니다</div>
<p>노라조</p>
<span>수퍼맨</span>
<p>이적</p>
<span>다행이다</span>
<p>현진영</p>
<span>Break me down</span>
```

$("div + p")는 div 바로 다음에 나오는 형제 수준의 p 요소, 즉, "노라조"를
$("div ~ p")는 div 요소 다음에 나오는 형제 요소들 중 모든 p 요소, 즉, "노라조", "이적", "현진영"과 일치된다는 것이죠.
다음은 어트리뷰트 필러의 예입니다.
a[title] : title 어트리뷰트를 갖는 하이퍼 링크와 일치된다.

a[href^="mailto:"] : href 값이 mailto 로 시작하는 하이퍼 링크와 일치된다.
a[href$=".pdf"] : pdf 파일에 링크가 걸린 모든 하이퍼링크와 일치된다.
a[href*="taeyo.net"] : taeyo.net 이라는 값이 포함되어 있는 모든 하이퍼 링크와 일치된다.
input[type="text"] : text 형식의 입력 컨트롤과 일치된다.
이러한 선택이 가능한 것은 시작부분(^) 혹은 끝부분($)을 가리키는 정규 표현식을 jQuery 가 지원하기 때문입니다.
jQuery 가 지원하는 CSS 셀렉터들

셀렉터	설명
*	모든 요소와 일치
E1	E1(태그명)인 모든 요소와 일치
E1.class	E1(태그명) 요소의 클래스가 class 와 동일한 요소와 일치
E1.#id	E1(태그명) 요소의 id 어트리뷰트 값이 id 와 동일한 요소와 일치
E1 E2	E1 요소의 자식인 모든 E2(태그명) 요소와 일치
E1 > E2	E1 요소 바로 아래 자식인 E2 요소와 일치
E1 + E2	E1 요소의 바로 다음에 나오는 형제요소 E2 와 일치
E1 ~ E2	E1 요소의 다음에 나오는 모든 형제 E2 와 일치
E1[*attr*]	*attr* 어트리뷰트를 갖는 E1 요소와 일치
E1[*attr*=*val*]	*attr* 어트리뷰트의 값이 *val*을 갖는 E1 요소와 일치
E1[*attr*^=*val*]	*attr* 어트리뷰트의 값이 *val* 값으로 시작하는 E1 요소와 일치
E1[*attr*$=*val*]	*attr* 어트리뷰트의 값이 *val* 값으로 끝나는 E1 요소와 일치
E1[*attr**=*val*]	*attr* 어트리뷰트의 값이 *val* 값을 포함하는 E1 요소와 일치

Default.htm

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title></title>
    <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
    <script>
        $(document).ready(
            function(){
                $("#song").css("border", "solid 1px silver");
                $("a[href^='mailto:']").css("background", "lightblue");
                $("input[type='button']").css("background", "yellow");

                $("div ~ b").css("background", "#efefef");
                $("div > b").css("border", "1px");
            });
    </script>
    <style type="text/css">
        * { font-size:12px; font-family:돋움 }
    </style>
</head>
<body>
    <div>
        <span><a href="http://taeyo.net">taeyo.net</a></span>
        <br />
        <a href="mailto:taeyo@A.net">태오의 메일주소</a>
        <p>
            <input type="button" value="클릭~" />
            <input type="checkbox" />
            <input type="radio" />
        </p>

        <div id="song">요즘 노래방에서 부르는 노래 목록입니다</div><br />
```

```
        <b>노라조</b>
        <span>수퍼맨</span><br />
        <b>이적</b>
        <span>다행이다</span><br />
        <b>현진영</b>
        <span>Break me down</span>
    </div>
</body>
</html>
```

일단, $(document).ready();라는 함수는 jQuery 가 제공하는 이벤트 메서드 중 하나인데요. 문서의 DOM 요소들을 조작 가능한 시점이 되면 자동으로 호출이 되는 이벤트 메서드라고 보시면 됩니다.
메서드의 인자로는 델리게이트 함수명을 기입하거나, 익명 함수를 작성하면 됩니다. jQuery 를 사용하는 경우에는 일반적으로 별도의 델리게이트 함수를 작성하지 않고, 익명함수를 그대로 작성하곤 합니다. 해서, 소스에서도 그렇게 처리한 것을 보실 수 있습니다.
그리고, 익명 함수 내에서는 다양한 jQuery 셀렉터들을 보실 수 있습니다. 일단, 셀렉터로 원하는 DOM 요소를 찾고, css() 라고 하는 jQuery 의 메서드를 사용해서 스타일을 매기는 것을 볼 수 있습니다. 기본적으로 DOM 요소는 css() 라는 메서드는 가지고 있지 않지만, jQuery 셀렉터가 반환하는 개체는 jQuery 확장 개체이기에, 이러한 메서드를 사용해서 쉽게 스타일을 매길 수가 있습니다.

2. jQuery 셀렉터 :필터사용하기

jQuery 셀렉터 : 기본 필터

기본적인 셀렉터에 더하여, jQuery 는 다양한 셀렉터 필터들도 제공을 합니다. 바로 앞에서 다루었던 어트리뷰트 셀렉터의 경우도 필터의 한 예입니다만, 어트리뷰트 셀렉터는 지금부터 다룰 필터들과는 표현식이 약간 달라서 기본 셀렉터와 함께 다루었습니다.

jQuery 는 위치를 기반으로 필터링 하거나, 하위 자식들을 필터링할 수 있게 한다거나, 컨텐트를 기반으로 필터링하는 등 다양한 방식으로 원하는 요소들을 필터링하여 선택할 수 있게 합니다. 그리고, 대부분의 필터는 : 을 시작문자로 사용하는 단어들입니다. 이러한 필터는 일반적으로 기본 셀렉터에 덧붙여 사용하곤 하지만, 단독으로 사용할 수도 있습니다. 그러면, 우선 가장 일반적인 필터들부터 차례대로 살펴보도록 할까요?

< 기본 필터 >

:first	선택된 개체들 중 첫 번째 요소와 일치합니다.
:last	선택된 개체들 중 마지막 요소와 일치합니다
:not(selector)	괄호에 주어진 셀렉터와 일치되는 모든 요소를 제외합니다.
:even	짝수 요소들과 일치합니다(0 부터 시작)
:odd	홀수 요소들과 일치합니다(0 부터 시작)
:eq(index)	인덱스에 해당하는 단일 요소와 일치합니다
:gt(index)	주어진 인덱스보다 높은 인덱스를 갖는 모든 요소와 일치합니다
:lt(index)	주어진 인덱스보다 낮은 인덱스를 갖는 모든 요소와 일치합니다
:header	모든 헤더 요소들(h1, h2, h3 등)과 일치합니다
:animated	현재 애니메이션이 동작중인 모든 요소와 일치합니다

각각의 필터는 셀렉터로서 독립적으로 사용될 수도 있고, 다른 셀렉터와 함께 사용할 수도 있습니다. 더불어, 여러 필터를 이어서 사용할 수도 있죠. 예를 들어 다음과 같이 말이죠.

tr:odd	문서에 나오는 모든 tr 중 홀수번째 tr 만을 선택합니다. 그런데, 인덱스는 0 부터 시작하므로, 사실, 제일 처음 나오는 tr 은 짝수 열로 인식된다는 부분에 주의하세요.
tr:gt(4)	문서에 나오는 tr 중 5 번째(인덱스 4) 후에 나오는 모든 tr 즉, 6 번째 tr 부터 모든 tr 을 선택합니다.
tr:last	문서에 나오는 모든 tr 중 가장 마지막에 존재하는 tr 만을 선택합니다.
:header:eq(1)	현재 문서에 나오는 모든 헤더(h1, h2, h3 등) 중에서 인덱스 위치가 1 인 헤더 즉, 눈에 보이기에 두 번째로 나오는 헤더를 선택합니다. 필터 앞에 아무것도 지정되지 않으면, 암시적으로 *이 지정된 것으로 인식됩니다. 즉, 이는 *:header:eq(1)와 동일한 표현으로 인식된다는 것이며, 페이지에 존재하는 모든 요소 중에서 헤더를 찾게 됩니다.

Filter01.htm

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title></title>
    <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
    <script type="text/javascript">
        $(document).ready(
            function() {
                $("tr:odd").css("background", "#efefef");
                $("tr:gt(4)").css("background", "silver");
                $("tr:last").css("background", "yellow");

                $(":header:eq(0)").css("font-weight", "bold").css("color", "blue");
            });
    </script>
```

```
    <style type="text/css">
        * { font-size:12px; font-family 돋움 }
        .nameTable { background: white; border-collapse:collapse }
    </style>
</head>
<body>
    <h3>수학 시험 점수</h3>
    <table class="nameTable" border="1" cellpadding="10">
        <tr><td width="100">name</td><td width="100">value</td></tr>
        <tr><td>Taeyo</td><td>80</td></tr>
        <tr><td>Dragon</td><td>98</td></tr>
        <tr><td>Eric</td><td>85</td></tr>
        <tr><td>Queeny</td><td>96</td></tr>
        <tr><td>Youngsun</td><td>88</td></tr>
        <tr><td>kobukii</td><td>90</td></tr>
        <tr><td>Mr.Net</td><td>92</td></tr>
    </table>
</body>
</html>
```

예제는 모든 홀수번째 tr(tr:odd)에게는 #efefef 배경색(아주 연한 회색)을 적용하고 있고요. 인덱스 번호 4 이후의 모든 tr 들(즉, 6 번째 tr 부터)에 대해서는 silver 배경색을 적용하고 있습니다. 그리고, 가장 마지막 tr(tr:last)에 대해서는 yellow 배경색을 주고 있고요. 더불어, 문서상에서 첫 번째로 나오는 헤더(:header:eq(0))에는 굵고 파란색의 폰트를 설정하고 있습니다.

매우 간단한 예제였지만, 이 예제는 마지막 셀렉터에 스타일을 주는 부분에서 재미있는 것이 하나 있는데요.

```
$(":header:eq(0)").css("font-weight", "bold").css("color", "blue");
```

라는 구문을 보시면, 셀렉터 대상에 대해서 css()라는 메서드를 연달아 사용하는 것을 볼 수 있죠? 이를 메서드 체인이라고 하는데요. jQuery 가 제공하는 모든 메서드는 그 반환값(return value)이 효과가 반영된 jQuery 개체이기 때문에 이런 식으로 메서드를 이어서 사용할 수가 있습니다. 다시 말해서, 저 표현은

```
var jQ = $(":header:eq(0)")
jQ = jQ.css("font-weight", "bold");
jQ = jQ.css("color", "blue");
```

와 동일하다 할 수 있다는 것이죠. 하지만, 이러한 방식은 직관적이기는 하지만 소스가 길어지는 단점이 있기에, 일반적으로는 메서드 체인으로 작성하곤 합니다.

자바스크립트는 줄바꿈을 인식하지 않기 때문에 다음과 같이 줄을 바꿔가면서 메서드 체인을 작성할 수 있습니다. 그리고, 대부분의 jQuery 개발자들도 다음과 같은 메서드 체인 방식을 즐겨 사용한답니다.

```
$(":header:eq(0)")
    .css("font-weight", "bold")
    .css("font-size", "12px")
    .css("color", "blue");
```

위의 메서드 체인에서는 css() 메서드를 3 번 연속 사용하고 있는데요. 이는 단지 메서드 체인의 예를 든 것임을 기억하세요. 실제로 저렇게 여러 스타일을 매기고자 하는 경우에는, css 메서드를 반복해서 사용하지 않고요. 스타일시트 파일에 필요한 css 클래스를 하나 만들어 두고, jQuery 의 addClass() 메서드를 사용하여 스타일을 한번에 지정하곤 합니다.

jQuery 셀렉터 : 컨텐트 필터와 자식 필터

이상 기본 필터에 대해서 알아봤는데요. 이 외에도 몇 개 안되지만 컨텐트 필터와 자식 필터라는 것이 또한 존재합니다. 이 또한 자주 쓰일만한 것이기에 마저 알아보도록 하죠.

먼저 컨텐트 필터입니다. 현재 버전에서는 꼴랑 다음과 같이 4 개의 필터가 존재합니다.

< 컨텐트 필터 >

:contains(text)	지정한 텍스트를 포함하는 요소들과 일치됩니다
:empty	자식을 가지지 않는 모든 요소와 일치됩니다. 더불어, 내부 텍스트를 가지지 않는 요소들도 이에 해당됩니다.
:has(selector)	지정된 셀렉터에 해당하는 요소를 갖는 모든 요소들과 일치됩니다.
:parent	부모인 모든 요소들과 일치됩니다. 자식 요소를 갖는 요소 뿐만 아니라 텍스트를 갖는 요소들이 이에 해당됩니다.

개인적으로는 :empty 필터가 상당히 유용했는데요. 예를 들면, 그리드 출력을 하는 경우에, 값이 누락되어 있는 셀들(td)을 찾아서 그 셀들만 배경색을 회색으로 처리한다거나 할 때는 완전 딱입니다. 물론, contains() 필터와 :has() 필터도 유용하긴 마찬가지 입니다.

b:contains('j')
div:has('ul')
table.nameTable td:empty
.nameTable td:contains('F')
table.nameTable tr:eq(0)

첫 번째 셀렉터는 "j"라는 단어를 포함하는 모든 b 요소들을 선택합니다.
두 번째 셀렉터는 모든 div 요소들 중에 ul 요소를 가지고 있는 div 만을 선택합니다.
세 번째 셀렉터는 nameTable 이라는 css 클래스가 지정된 table 들 중에서 자식 요소인 td 내부에 텍스트 값이 없는(또는 자식요소가 없는) 모든 td 를 선택합니다.
네 번째 셀렉터는 nameTable 이라는 css 클래스가 지정된 요소들 중에서 그 내부의 td 가 텍스트로 "F"란 단어를 포함하고 있는 모든 td 를 선택합니다.
마지막 셀렉터는 nameTable 이라는 css 클래스가 지정된 table 내의 tr 중에서 첫번째 tr 을 선택합니다.

만일, 문서 내에 table 요소가 한 개만 존재하고, 그 table 에 nameTable 이라는 css 클래스가 적용이 되었다면, 다음의 셀렉터는 모두 같은 것을 선택한다는 것!
$("table.nameTable tr")
$(".nameTable tr")
$("table tr")
$("tr")
네. 당연한 이야기입니다. 굳이 이러한 이야기를 꺼낸 것은 셀렉터를 처음 접할 경우에는 셀렉터를 어떻게 작성하는 것이 좋은지에 대해 자꾸 걱정하는 분들이 있어서요. 셀렉터는 문서 내에서 원하는 요소를 명확히 찾아낼 수 있는 직관적인 구문이면 뭐든지 오케이입니다. 특별히 어떤 구문이 더 좋다라는 식의 정해진 룰은 없습니다. 물론, 자꾸 사용하시다보면 자신만의 패턴이 생기긴 할 겁니다~(저의 경우는 가급적 구문을 상세하게 작성하는 것을 권합니다)

그러면, 이제 예제를 통해서 컨텐트 필터들의 사용 예를 한번 살펴볼까요?
Filter02.htm

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title></title>
    <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
    <script type="text/javascript">
        $(document).ready(
            function() {
                $("b:contains('j')").css("background", "yellow");
                $("div:has('ul')").css("border", "solid 1px green");

                $("table.nameTable td:empty").css("background", "#dddddd");
                $(".nameTable td:contains('F')").parent().css("background", "pink");
                $("table.nameTable tr:eq(0)").css("background", "lightblue");
            });
    </script>
    <style type="text/css">
        * { font-size:12px; font-family 돋움 }
        .nameTable { background: white; border-collapse:collapse }
    </style>
</head>
<body>
    <h3>jQuery 란?</h3>
    <div>
        <b>jQuery</b>는 가볍고 빠르며, 간결한 오픈소스 스크립트 라이브러리입니다.
        이를 이용하면 Rich 웹 UI 를 개발하는 데 도움이 되는 다양한 기능들 즉,
        <b>HTML 문서 트래버스</b>, <b>이벤트 처리</b>, <b>애니메이션</b>, <b>Ajax 상호작용
        </b>등을 지원하여 빠르고 견고하게 리치 웹 애플리케이션 개발을 할 수 있도록 지원합니다.
    </div>
    <br />
    <div>
        이러한 <b>jQuery</b>의 기능적인 특징은 다음과 같습니다
    </div>
    <br />
    <div>
        <ul>
            <li>막강한 CSS 셀렉터</li>
            <li>크로스 브라우저 지원</li>
            <li>메서드 체인</li>
```

```
            <li>Ajax 지원</li>
            <li>풍부한 플러그 인 지원</li>
        </ul>
    </div>
    <br />
    <table class="nameTable" border="1" cellpadding="3">
        <tr><td width="100">이름</td><td width="100">성별</td><td width="100">기타 정보</td></tr>
        <tr><td>Taeyo</td><td>M</td><td>...</td></tr>
        <tr><td>Dragon</td><td>M</td><td>...</td></tr>
        <tr><td>Eric</td><td>M</td><td>...</td></tr>
        <tr><td>Queeny</td><td>F</td><td>...</td></tr>
        <tr><td>Youngsun</td><td></td><td>...</td></tr>
        <tr><td>Hera</td><td>M</td><td>...</td></tr>
        <tr><td>Secret</td><td>F</td><td>...</td></tr>
    </table>
</body>
</html>
```

셀렉터들이 어떤 요소를 선택하고, 어떻게 동작할지는 소스를 보시면 이해가 되실 것입니다. 예제에서 사용한 셀렉터는 이미 예제에 앞서 설명을 드린 셀렉터 그대로이니까요. 다만, 4 번째 구문인 다음 구문에서는 parent()라는 처음 보는 메서드가 하나 사용된 것을 보실 수 있을 겁니다.

```
$(".nameTable td:contains('F')").parent().css("background", "pink");
```

parent()라는 메서드는 말 그대로 현재 셀렉터에 의해 선택된 개체 집합의 바로 위 부모 개체를 반환하는 메서드입니다. 고로, 이 구문은 .nameTable 이라는 css 클래스를 갖는 개체의 td 들 중에서 "F"라는 텍스트를 포함하는 모든 td 를 찾아낸 다음(여기까지가 셀렉터), 그의 부모인 tr 개체를 얻어내서(parent 메서드), 그의 배경색을 핑크빛으로 물들이는 것입니다.

예상대로, b 요소 중 "j"라는 단어가 포함된 것들은 노란 배경색으로 출력되고 있으며, div 요소들 중에서 ul 이라는 자식 요소를 가진 개체는 녹색 테두리로 나타나고 있습니다. 또한, 테이블에서 내용이 없는 td 는 연한 회색으로 채워져 있으며, td 안의 텍스트가 "F"라는 글자를 포함하고 있으면 그의 부모인 tr 이 핑크색으로 출력되는 것을 볼 수 있습니다. 마지막으로, 가장 첫 번째 tr 은 연한 파란색으로 출력되고 있고요.

컨텐트 필터는 이처럼 요소의 컨텐트와 관련하여 필터링을 하는데 사용할 수 있는 기능입니다.
컨텐트 필터 외에 또 다른 필터로 자식 필터라는 것도 있습니다. 자식 필터는 말 그대로 현재 요소의 자식들에 대한 필터를 할 수 있는 기능을 제공하는데요. 현재 버전의 jQuery(1.3.2)에서는 다음과 같이 4 개의 자식 필터가 제공됩니다.

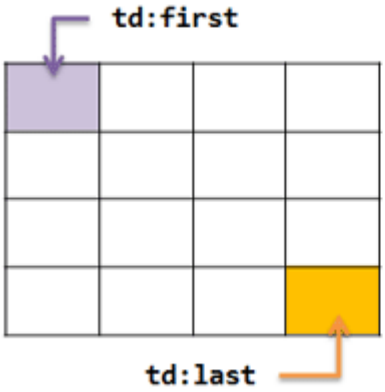
< 자식 필터 >

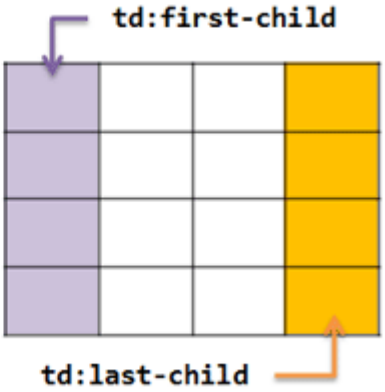
:nth-child(index/even/odd)	자식 중 index 로 지정된 위치의 요소들과 일치되거나, even, odd 에 해당하는 자식들과 일치됩니다. 단, 여기서의 index 는 1 부터 시작합니다.
:first-child	첫 번째 자식인 모든 요소와 일치됩니다.
:last-child	마지막 자식인 모든 요소와 일치됩니다.
:only-child	자신이 부모 요소의 유일한 자식인 모든 요소와 일치됩니다

처음 이 필터를 접할 경우에는 이전에 다루었던 기본 필터 중 :first, :last 와 유사한 필터가 아닐까 하고 생각할 수 있는데요. 사실 이들은 기존의 필터와 상당한 차이가 있습니다.
:first, :last 필터가 단일 요소를 선택하는 반면, 자식 필터들은 단일 요소가 아닌 해당 요소들의 집합을 선택한다는 것이 차이입니다. 예를 들어, 다음 두 셀렉터의 차이를 한번 생각해 볼까요?

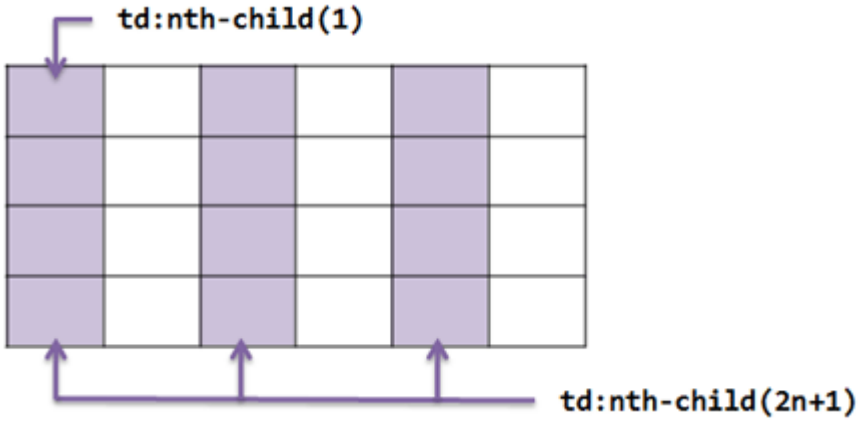
```
td:first
td:first-child
```

td:first 는 현재 테이블 안에서 첫 번째로 나오는 td 하나 만을 선택하는 반면, td:first-child 는 현재 테이블 안에서 나오는 첫 번째 수준의 td 들을 모두 선택한다는 것이죠





차이가 느껴지시죠?
:nth-child 라는 필터도 동작 방식은 동일합니다만, 인덱스를 지정할 수 있거나, even 이나 odd 와 같은 값을 지정하여, 짝수나 홀수 번째 그룹이 선택되도록 할 수 있습니다. 심지어는 Xn+Y 와 같은 수식을 사용해서 특정 규칙에 따라 선택할 수도 있습니다. 예를 들어, td:nth-child(3n)과 같이 작성하게 되면, 이는 3 의 배수에 해당하는 위치의 td 그룹이 선택될 것이며, td:nth-child(2n+1)과 같이 작성하게 되면, 2 의 배수에 1 을 더한 값에 해당하는 위치의 td 그룹이 선택될 것입니다. 수식에서의 n 은 0 부터 계산되기에, 3n 은 0,3,6,9 .. 에 해당하는 그룹이 되며, 2n+1 은 1,3,5,7… 에 해당하는 인덱스 그룹이 선택될 것입니다.
다만, 여기서의 인덱스는 0 부터 시작하는 것이 아니라 1 부터라는 것에 유념하셔야 합니다. 요 부분이 살짝 짜증(?) 나는 부분이긴 한데요. 앞서 다룬 기본 필터(:even, :odd, :eq(index), :gt(index) 등)에서는 인덱스가 0 부터 시작했으나, 자식 필터인 :nth-child 에서의 인덱스는 1 부터 적용된다는 것이죠.
즉, :first-child 와 동일한 것은 :nth-child(1)이지, :nth-child(0)이 아니라는 것입니다. 자식 필터에서 :nth-child(0)은 그 어떤 것과도 일치되지 않습니다. **꼭 기억하세요. 자식 필터를 사용하는 경우에는 인덱스가 1 부터 시작한다는 것을요.**



3. jQuery :폼필터및조작메서드(each 등)

이번 강좌는 셀렉터의 마지막 강좌로 폼과 관련된 셀렉터에 대해서 알아볼까 합니다. 사실상, 웹 페이지에서 서버로 데이터를 전송하려면 입력 UI 로 다양한 폼 컨트롤들을 사용할 수 밖에 없는데요

폼 필터들에는 어떤 것들이 있는지 일단 정리한 것을 먼저 보고 이야기를 계속해 나가보겠습니다.

:input	모든 input, textarea, select, button 요소들과 일치됩니다
:text	text 타입의 모든 input 요소들과 일치됩니다.
:password	password 타입의 모든 input 요소들과 일치됩니다.
:radio	radio 타입의 모든 input 요소들과 일치됩니다.
:checkbox	checkbox 타입의 모든 input 요소들과 일치됩니다.
:submit	submit 타입의 모든 input 요소들과 일치됩니다.
:image	image 타입의 모든 input 요소들과 일치됩니다.
:reset	reset 타입의 모든 input 요소들과 일치됩니다.
:button	모든 button 요소들 및 button 타입의 input 요소들과 일치됩니다.
:file	file 타입의 모든 input 요소들과 일치됩니다.
:hidden	hidden 상태인 모든 요소들과 일치됩니다.

이미 필터에 익숙한 여러분이기에 보면서 이해가 되실텐데요. 보이는 것처럼 모든 입력 컨트롤을 선택하고 싶다면 $(":input") 라고 작성하면 되고, 체크박스만 선택하고 싶다면 $(":checkbox")라고 jQuery 를 작성하면 됩니다. 히든 필드는 $(":hidden") 이라고 작성해서 선택할 수 있고요. 하지만, 생각 외로 이 폼 필터들을 자주 사용할 일은 없는 편인데요(물론, 사용하기 나름이겠지만 저의 경우는 그다지 사용할 일이 많지 않았습니다).

반면, 체크박스 중 체크된 것들을 모두 알아온다던가, 셀렉트 컨트롤들 중에서 선택된 것들을 알아낸다던가 하는 작업은 상대적으로 자주 요구되는 일 중에 하나입니다. 예를 들어, 사용자가 체크한 값들이나 선택한 값들을 화면에 출력해주는 일을 생각해 볼 수 있습니다. 다음의 코드를 한번 볼까요?

```
var sports = $("input:checkbox");
for (var i = 0; i < sports.length; i++) {
  if (sports[i].checked == true) {
    //두 썸띵!
  }
}
```

var sports = $("input:checkbox"); 와 같은 셀렉터 혹은 셀렉터 필터에 의해 반환하는 개체는 Html 요소들의 배열입니다. 즉, sports 라는 변수는 Html 요소들의 배열이지, jQuery 개체의 배열이 아니라는 것이죠.

상기 코드는 현재 페이지에 존재하는 모든 체크박스들을 일일히 확인해가면서 선택된 것을 찾아내는 코드입니다만, jQuery 의 필터 중에는 이와 같은 작업을 보다 쉽게 해주는 필터들도 있습니다. 바로 다음과 같은 것들이죠.

:enabled	현재 enable 상태인 모든 요소와 일치됩니다.
:disabled	현재 disable 상태인 모든 요소와 일치됩니다.
:checked	체크된 모든 요소들과 일치됩니다.
:selected	선택된 모든 요소들과 일치됩니다.

예를 들면, :checked 와 같은 필터는 폼에서 체크된 모든 요소들을 선택하게 됩니다. 즉, 이 필터를 사용하면 모든 체크박스를 일일히 확인할 필요없이, 체크가 된 체크박스들만을 얻어올 수 있다는 것이죠. 해서, 이 필터를 사용하면 이전의 체크박스 확인 코드가 다음과 같이 좀 더 간단해 질 수 있습니다.

```
var sports = $("input:checked");
var result = "";
for (var i = 0; i < sports.length; i++) {
  //두 썸띵!
}
```

자. 그럼 간단한 예제를 통해서 확인을 해볼까요?

```
<html>
<head>
    <title></title>
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
    <script type="text/javascript">
        $(document).ready(function() {

            $("#DoIt").click(function() {
```

```
                var sports = $("input:checked");
                var result = "";
                for (var i = 0; i < sports.length; i++) {
                        result += $(sports[i]).next().text() + ",";
                }
                $("#result").text(result);
            });
});
    </script>
</head>
<body>
    <p>
        스포츠 : <input id="c1" type="checkbox" /><label for="c1">농구</label>
        <input id="c2" type="checkbox" /><label for="c2">배구</label>
        <input id="c3" type="checkbox" /><label for="c3">축구</label>
        <input id="c4" type="checkbox" /><label for="c4">야구</label>
    </p>
<p>
        <input id="DoIt" type="button" value="선택" />
</p>
    <p>
        당신이 선택하신 스포츠는 <span id="result"></span> 입니다.
    </p>
</body>
</html>
```

화면에는 4 개의 체크박스가 존재하고 있고, 1 개의 버튼과 1 개의 출력용 span 구역이 정의되어 있습니다. 그리고, 구현하려는 기능은 사용자가 몇몇 체크박스에 체크를 하고, "선택" 버튼을 클릭하면, span 영역에 그 선택된 값들을 출력하려는 것입니다. 하지만, 스크립트 구역의 소스는 생각보다 설명을 드릴 것이 조금 많네요. 우선 이벤트에 관한 것부터 설명 드리겠습니다.

우선, $(document).ready()라는 이벤트 메서드를 보세요. 메서드 인자로는 함수를 작성한다고 말씀을 드렸었죠. 즉, 다음과 같이 말입니다.

```
$(document).ready(pageLoad);
function pageLoad()
{
   //.. 페이지 로드시에 할 일
}
```

하지만, 일반적으로는 저렇게 굳이 함수를 분리해서 작성할 이유가 없어서, pageLoad 라는 함수를 별도로 작성하지 않고, 그냥 다음과 같이 익명 함수로 작성하는 편입니다.

```
$(document).ready(
   function()
   {
      //.. 페이지 로드시에 할 일
   }
);
```

그리고, 대부분은 가독성을 위해서 다음과 같이 붙여서 쓰곤 하는 편이죠.

```
$(document).ready(function(){
    //.. 페이지 로드시에 할 일
});
```

그 이후 등장하는 소스를 계속해서 보시면, 페이지가 준비가 되는대로(즉, ready() 함수 안에서) 다음과 같은 코드를 사용해서 버튼에 click 이벤트를 걸어주는 것을 볼 수 있습니다.

```
$("#DoIt").click( someFunction );
```

이는 #DoIt 이라는 개체에게 동적으로 클릭 이벤트를 걸어주는 jQuery 코드입니다. 사실, 이는 document.getElementById("DoIt").click = someFunction; 이라는 자바스크립트 함수와 같은 것이라 볼 수 있죠. 하지만, 코드가 간결하고, 메서드 체인을 통해서 여러 작업을 이어서 지정할 수 있다는 장점이 있습니다. 예를 들면, 다음과 같이 말입니다.

```
$("#DoIt").click( someFunction ).mouseover( otherFunction ).css("color", "red");
```

즉, 이와 같은 식으로 작성해서, 클릭 이벤트와 마우스 오버 이벤트를 걸어줌과 동시에 color 스타일 값도 red 로 변경하는 작업을 한번에 할 수 있다는 것이죠.

사실, 오리지날 jQuery 에서의 이벤트 설정 작업은 상기와 같은 코드를 사용하는 것이 아니라 다음과 같이 bind 라는 메서드를 통해서 설정해야 합니다.

```
$("#DoIt").bind("click", someFunction );
```

하지만, jQuery 는 자주 사용하는 몇몇 이벤트(몇몇이라고 하기엔 꽤 많지만)에 대해서는 click 과 같이 단축 메서드를 작성해 놓았기에, bind 대신 click 과 같이 직관적인 메서드를 사용해서도 이벤트 핸들러를 걸 수가 있는 것입니다.

다시 예제에 집중하도록 하죠.

```
$("#DoIt2").click(function() {
  //.. 클릭 시 해야할 작업
});
```

click 메서드의 인자로 지정된 익명함수 내에서는 원래의 개체에 대한 컨텍스트가 유지되기 때문에, 함수 내에서 this 라는 키워드를 사용할 수 있으며, 이 this 는 Doit 이라는 html 요소를 의미합니다. 만일, jQuery 개체로 다시 감싸고 싶다면, 함수 내부에서 $(this)와 같이 작성하시면 되지요.
자. 그럼 이제 클릭 이벤트 핸들러 안에다 어떤 코드를 작성했는지 한번 살펴보도록 하겠습니다. 단, 그 전에 이제부터 작성하는 코드들은 "버튼이 클릭되는 경우"에만 수행된다는 것을 기억하세요.

```
var sports = $("input:checked");
var result = "";
for (var i = 0; i < sports.length; i++) {
    result += $(sports[i]).next().text() + ",";
}
$("#result").text(result);
```

자. 이 코드는 앞서 설명한 :checked 필터를 사용하여 체크가 된 체크박스들을 얻어오고 있고요. 그를 sports 라는 변수에 담고 있습니다. 결국, sports 라는 변수는 셀렉터에 의해 필터링된 html 요소들의 배열이라고 볼 수 있죠. 그들을 루프를 돌면서, 각 체크박스의 다음에 있는 html 요소로 접근한 뒤(next() 메서드), 그 요소의 텍스트값을 얻어와서 문자열로 구성하고 있습니다.
여기서 우리가 추가로 배우게 되는 2 개의 jQuery 메서드가 있는데요. 하나는 next() 이고, 다른 하나는 text() 입니다. next()란 메서드는 현재의 개체 요소와 형제 수준인 요소 중 바로 다음에 나오는 개체를 얻어오는 메서드입니다. 현재 예제는 각각의 체크박스 바로 옆에 <label> 개체들이 놓여있는 것을 보실 수 있을 겁니다. 고로, $(sports[i]).next() 라는 표현은 현재의 체크박스 바로 옆에 있는 label 개체를 찾아가게 되는 것이죠. 단, 루프 안에서 sports[i].next() 라고 코드를 작성한 것이 아니라, $(sports[i]).next()라고 작성한 것에 주목하셔야 하는데요. sports 라는 배열에 들어있는 개체들은 jQuery 개체가 아닌 html 요소일 수 있기 때문입니다. 해서, 이를 $()를 사용하여 확실하게 jQuery 개체로 포장해야만 합니다. jQuery 개체가 아니면 next() 메서드를 사용할 수 없으니까요.

text() 메서드는 해당 개체가 가지고 있는 컨텐트를 텍스트로 반환하는 메서드입니다. 이와 유사한 메서드로는 html()이 있는데요. 이는 개체가 포함하고 있는 html 컨텐트를 반환하는 메서드입니다. 이 메서드들은 인자를 사용하지 않으면 값을 가져오는 역할을 하지만, 인자로 값을 지정하면 그 값으로 원래의 값을 바꾸는 역할을 합니다. text() 메서드의 재미있는 점은 개체의 컨텐트가 html 요소들을 포함하고 있다고 해도, 다 무시하고 텍스트만을 반환한다는 점입니다. 즉, taeyo 와 같은 span 개체가 있을 때, 이 span 개체에 대해서 text()를 수행하면, 반환값은 내부 html 을 제외한 "taeyo"가 되고요. html() 메서드를 사용하면 taeyo를 반환한다는 것이죠.
그리하여, 예제에서의 result += $(sports[i]).next().text() + ","; 라는 구문은 현재 루프 안에서, 체크박스 개체의 바로 옆에 있는 개체(label)의 텍스트를 읽어와서 그 텍스트에 ","라는 문자를 더한 다음, 이를 result 라는 변수에 덧붙이는 역할을 하게 됩니다. 즉, 사용자가 체크한 체크박스의 명칭을 차곡차곡 문자열에 더하는 것이죠.
그 후, 예제는 그렇게 만들어진 문자열 result 를 span 요소에 출력하는 것을 볼 수 있습니다. 텍스트 설정을 위해서 text() 메서드를 사용하고 있는 것도 잊지말고 짜려봐 주세요.

이야기를 진행하다 보니, 원래의 폼 관련 셀렉터에 대한 설명보다는 jQuery 개체가 제공하는 트래버싱(traversing, 탐색) 메서드나 조작(Manipulation) 메서드에 대해 설명을 하게 되었네요.

그렇다면, 이야기를 꺼낸 김에 루프를 좀 더 편하게 작성할 수 있는 jQuery 의 기본 제공 메서드인 each() 메서드에 대해서도 설명을 좀 드리고 넘어갈까 합니다.
each() 메서드는 jQuery 개체가 제공하는 메서드로서, 모든 일치된 개체들에 대해서 순차적으로 특정 함수를 수행할 수 있게 해주는 메서드입니다. 문법은 다음과 같아요.

```
jQuery(selector).each( function );
```

each() 메서드 안에 인자로 사용하는 것은 역시나 또 함수입니다. 그리고, 역시나 주로 익명 함수를 사용하게 되죠. 해서, 일반적으로 다음과 같은 구문을 가지게 됩니다.

```
jQuery(selector).each( function(i) {
    //.. 루프 안에서 할 일을 작성
});
```

익명 함수의 i 라는 인자는 0 부터 시작하는 인덱스 값이 넘어옵니다. 마치 for 문에서의 i 와 동일하다고 보시면 됩니다만, 필요치 않다면, 인자를 지정하지 않으셔도 됩니다.

each() 메서드는 상당히 자주 사용하게 되는 메서드입니다. 제가 샘플로 만들어 놓은 jQueryBoard 에서도 이는 예외가 아닙니다. 이는 셀렉터에 의해 추출된 개체들에 대해서 각각 지정한 함수를 수행하게 합니다. 해서, 루프를 사용할 때와 마찬가지로 각 요소에 대해 어떤 작업을 하고 싶다면 바로 그 함수 안에다가 수행할 내용을 작성하면 됩니다.

그러면, 이번 예제의 for 루프문을 each() 메서드를 사용해서 바꾸어 볼까요?

```
var result = "";
$("input:checked").each(function(i) {
    result += $(this).next().text() + ",";
```

```
});
$("#result").text(result);
```

만일, 2 개의 체크박스가 체크되었다면, each() 안의 함수는 그 2 개의 체크박스에 대해 각각 호출될 것이고, 구문 내 this 는 각각의 호출 시에 그에 해당하는 체크박스가 될 것입니다.

다음 구문을 보시죠.

```
$("#DoIt").click(function() {
    var result = "";
    $("input:checked")
        .each(function(i) {
            result += $(this).next().text() + ",";
         })
        .css("background", "yellow");

    $("#result").text(result);
});
```

each() 메서드에 이어서 css() 메서드를 체인으로 연결해서 작성하고 있습니다. 이렇게 작성하면, 체크된 모든 체크박스에 대해서 each() 메서드를 사용한 기존 작업을 수행할 뿐 아니라, 배경색을 노란색으로 지정하는 것까지 수행하게 됩니다.

4. jQuery :조작(Manipulation) 메서드

이번 강좌에서는 이전에 언급한대로 jQuery 의 조작(Manipulation) 관련 메서드들에 대해서 다루어 볼까 합니다. 사실, 이미 기초를 뗀 여러분들에게 조작관련 메서드, 탐색관련 메서드를 굳이 구분지어서 설명할 필요는 없다는 느낌도 들긴 하지만 그래도 깔끔한 정리를 위해서 말이죠 ^^;
조작 관련 메서드라 함은 요소에 값을 지정한다거나, 특정 요소의 값을 읽어온다거나 하는 작업을 포함해서, 동적으로 요소 자체를 생성, 삭제, 복사, 제거하는 기능들을 의미합니다.

우선, 현재 jQuery 에서 제공하는 조작과 관련한 메서드들은 크게 7 개의 카테고리로 나누어져 있는데요. 그들은 각각 요소의 내용을 조작하는 메서드, 요소를 추가하거나 삭제하는 메서드, 요소를 래핑하거나 바꿔치기 하는 메서드, 요소를 제거하거나 복사하는 메서드들로 구분되어 있습니다
제가 상대적으로 자주썼던 5 개의 카테고리만을 설명할 예정입니다.
jQuery 의 조작 메서드 중 하나의 카테고리인 요소의 내용을 다루는 메소드로는 다음과 같은 것들이 있습니다.

내용 변경 메서드	
html()	일치된 요소의 html 내용을 가져옵니다. 이는 요소의 innerHTML 값과 동일합니다. 만일, 일치된 요소가 여러 개라면 그 중 첫 번째 요소의 HTML 을 가져옵니다.
html(*val*)	일치된 요소의 html 본문을 *val* 값으로 설정합니다. 만일, 일치된 요소가 여러 개라면 모든 요소에 이러한 작업을 수행합니다
text()	일치된 모든 요소의 텍스트를 합쳐서 가져옵니다.
text(*val*)	모든 일치된 요소의 텍스트를 *val* 값으로 설정합니다.

사실, 이 메서드들은 이전 강좌에서도 만나보았던 것인데요.
text() 메서드의 재미있는 점은 개체의 컨텐트가 html 요소들을 포함하고 있다고 해도, 다 무시하고 텍스트만을 반환한다는 점입니다. 즉, taeyo 와 같은 span 개체가 있을 때, 이 span 개체에 대해서 text()를 수행하면, 반환값은 내부 html 을 제외한 "taeyo"가 되고요. html() 메서드를 사용하면 taeyo를 반환한다는 것이죠.'

jQuery 로 검색된 대상 요소가 여러 개일 경우, text() 메서드와 html() 메서드의 동작이 약간 다르다는 것인데요. 일치된 요소가 여러 개인 경우, text() 메서드는 일치된 모든 대상의 텍스트를 결합해서 반환하는 반면, html() 메서드는 일치된 요소들 중 첫 번째 요소의 html 만을 반환한다는 것이 그 차이이지요. 크게 중요한 사항은 아니지만, 이 차이를 모르면 jQuery 를 사용하다가 간혹 결과가 이상하게 나온다고 불만을 가질 수 있으므로 기억 해 두시기 바랍니다.

이어지는 메서드들은 특정 요소나 개체 집합을 다른 요소의 "내부"에다가 앞, 뒤로 덧붙이는 기능을 제공하는 메서드들입니다.

추가(요소 내부에) 관련 메서드	
append(*content*)	일치된 요소의 내부에 *content* 를 덧붙입니다.
appendTo(*selector*)	일치된 요소를 *selector* 에 의해 일치된 모든 요소들 내부에 덧붙입니다. 만일, 일치된 요소가 본문에 존재하는 개체(예, $("#some")과 같은)라면 그를 제거한 후 복사(즉, 이동)합니다. 설명이 어려우면 밑의 예제를 참고하세요.
prepend(*content*)	append(*content*)와 동일하며, 다만, 내부 앞쪽에 붙여 넣습니다.
prependTo(*selector*)	appendTo(*selector*)와 동일하며, 다만, 내부 앞쪽에 붙여 넣습니다.

append(content) 메서드는 이름이 의미하는 바가 명확하기에, 쉽게 이해할 수 있을 것입니다. 인자로 지정된 content 를 jQuery 개체 내부에 덧붙이는 것이니까요. 예를 들어, 다음과 같은 표현이 있다고 가정해 봐요.

```
$("b.link").append("(클릭)");
```

이는 link 라는 css 클래스를 갖는 B 태그 내부에 "(클릭)" 이라는 문자열을 추가하는 예입니다. B 태그 내부 뒤쪽에 덧붙이는 것이기에 "(클릭)"이라는 단어도 당연히 굵게 표현될 것입니다. 즉, 다음과 같은 최종 html 이 형성된다는 것이죠.

```
<b class="link">굵은 글자임 (클릭)</b>
```

이 메서드는 상당히 유용하게 사용되곤 합니다. 주석 표시를 HTML 로 표현하는 경우에는 <sup>라는 태그를 사용하곤 하는데요. 그것을 수동으로 일일이 태그를 달자면 상당히 노가다스럽습니다. 주석 태그를 수십개 달아뒀는데, 순서라도 바뀌는 날에는 생각만해도 피곤해 집니다. 하지만, 주석을 표현해야 할 단어들에 대해서 .annotation 이라는 스타일 클래스를 지정해 두었다면(대부분 스타일을 지정하죠), 다음과 같은 jQuery 구문을 통해서 주석표시를 일괄적으로 해결할 수 있습니다.

```
$(".annotation").each(function(i) {
  var idx = i + 1;
  $(this).append("<sup>" + idx + "</sup>");
});
```

이는 .annotation 라는 css 클래스가 지정된 html 요소들에 대해서 ¹, ²와 같은 주석 표현 태그를 동적으로 덧붙여주는 코드입니다. 일일이 단어의 뒷 부분에 ¹와 같은 태그를 수동으로 추가할 필요없이, jQuery 를 이용해서 동적으로 그러한 표현을 추가할 수 있는 아주 유용한 예라고 볼 수 있습니다.
다음 그림은 그렇게 처리한 예를 보여줍니다.

jQuery란?

jQuery는 가볍고 빠르며, 간결한 오픈소스[1] 스크립트 라이브러리입니다. 이를 이용하면 Rich 웹 UI를 개발하는 데 도움이 되는 다양한 기능들 즉, HTML 문서 트래버스[2], 이벤트 처리, 애니메이션, Ajax[3] 상호 작용 등을 지원하여 빠르고 견고하게 리치 웹 애플리케이션 개발을 할 수 있도록 지원합니다.

append(content) 메서드가 간단하면서도 유용하기에, 그와 유사한 appendTo(selector) 메서드도 관심을 가져 볼만 합니다. 그런데, 이 메서드는 사용법이 상대적으로 살짝 복잡하게 느껴질 수 있습니다. append 와는 덧붙이기의 대상이 반대이며, 더불어, 셀렉터에 의해 일치된 대상 모두에게 덧붙여지기 때문이지요. 다음 예를 한번 볼까요?

```
$("#linkText").appendTo("a.link");
```

이는 #linkText 라는 아이디를 갖는 요소를 얻어서, 그를 link 라는 css 클래스를 갖는 "모든" 하이퍼링크 뒤에 덧붙이게 됩니다. 그러면서, 원래의 #linkText 요소는 사라집니다. 원본 개체는 복사되면서 제거되는 것이죠. 더불어, 다음과 같은 구문도 가능합니다.

```
$("<font>(클릭)</font>").appendTo("b");
```

이는 (클릭)라는 태그 문자열을 동적으로 생성한 뒤, 그를 모든 B 태그의 뒤에 덧붙이는 역할을 합니다.

말이 나온 김에, 우리는 $()라는 핵심 표현에 대해 살펴보고 넘어가야 할 필요성이 있습니다.
여러분은 $() 표현이 jQuery()라는 구문의 단축표현이자, 셀렉터를 수행하기 위한 구문이라고 알고 있을 것입니다만, 실은 그 외에도 3 가지의 기능을 더 가지고 있습니다.
간략하게 설명을 드리면 각 기능은 이렇습니다.

1. 인자로 셀렉터를 지정한 경우에는 일치되는 모든 요소를 찾는다.
2. 인자로 html 태그를 지정한 경우에는 동적으로 그 요소를 생성한다.
3. 인자로 특정 DOM 요소를 지정하면, 그 요소의 jQuery 래퍼를 생성한다.
4. 인자로 function 을 지정하면, 그는 $(document).ready() 메서드와 동일하다.

해서, $("(클릭)") 라는 표현은 동적으로 font 요소를 생성하게 되는 것이고요. $("#linkText")라는 표현은 #linkText 라는 셀렉터에 일치하는 요소들을 검색하게 되는 것이죠.

prepend 와 prependTo 메서드도 쉽게 이해하실 수 있을 것입니다. 단지, pre-append 라는 의미라고 생각하시면 됩니다. 즉, prepend 는 append 와 동일하지만, 요소를 뒤에 덧붙이는 것이 아니라 앞에 덧붙인다는 차이가 있습니다. prependTo 도 마찬가지고요.

세번째 메서드 그룹은 위의 추가(내부) 관련 메서드와 유사하긴 하지만, 요소 내부에 추가를 하는 것이 아니라, 요소 외부에 추가하는 메서드 그룹입니다.

추가(요소 외부에) 관련 메서드	
after(*content*)	일치된 요소 뒤에 *content*를 삽입합니다. 요소 내부가 아닌 외부에 삽입된다는 것을 제외하면 append 와 동일합니다.
before(*content*)	일치된 요소 앞에 *content*를 삽입합니다. 요소 내부가 아닌 외부에 삽입된다는 것을 제외하면 prepend 와 동일합니다.
insertAfter(*selector*)	일치된 요소를 *selector*에 의해 일치된 모든 요소들 뒤쪽에 삽입합니다. 요소 내부가 아닌 외부에 삽입된다는 것을 제외하면 appendTo 와 동일합니다.
insertBefore(*selector*)	insertBefore(*selector*)와 유사하나, 요소 앞쪽에 삽입합니다. 요소 내부가 아닌 외부에 삽입된다는 것을 제외하면 prependTo 와 동일합니다.

after 메서드는 앞서 설명한 append 메서드와 눈에 보이는 결과는 유사합니다. 다만, 태그 내부에 덧붙이는 것이 아니라, 태그 바깥쪽에 덧붙인다는 차이가 있지요. 예를 들어, 다음 구문을 보도록 해요.

```
$("a").after("<font>(클릭)</font>");
```

이는 모든 하이퍼링크 바로 뒤에 "(클릭)" 이라는 태그 문자열을 덧붙이는 구문입니다. 해서, 결과로 다음과 같은 태그가 구성되는 것이죠.

```
<a href="http://taeyo.net">태오 사이트</a><font>(클릭)</font>
```

만일, after 메서드 대신 append 구문을 다음과 같이 사용했다면,

```
$("a").append("<font>(클릭)</font>");
```

이는 다음과 같은 결과 태그를 구성했을 것입니다.

```
<a href="http://taeyo.net">태오 사이트<font>(클릭)</font></a>
```

같은 방식으로 before 메서드도 prepend 메서드와 유사합니다. 태그 내부에 삽입 되느냐 외부에 삽입되느냐만 차이가 있는 것이죠. 마찬가지로, insertAfter()는 appendTo() 메서드와 동일하게 동작하며, insertBefore()는 prependTo()와 동일하게 동작합니다. 다만, 자식 요소로서 태그 내부에 삽입되느냐, 형제 요소로서 외부에 삽입되느냐의 차이만 있다는 것이죠.

스피디하게, 이어지는 소개할 메서드 그룹은 삭제용 메서드와 복사용 메서드들입니다.

삭제 메서드	
empty()	모든 일치된 요소들의 자식 노드들을 제거합니다.
remove()	모든 일치된 요소들을 DOM 에서 제거합니다.
복사 메서드	
clone()	일치된 요소를 복사하고, 그를 선택합니다.
clone(bool)	이벤트 처리기를 포함하여 DOM 요소를 복사하고 그를 선택합니다.

별도의 설명이 필요없을 정도로 직관적이면서도, 매우 자주 사용하게 되는 메서드들입니다. 다만, 복사 관련 메서드(clone)는 약간의 부연 설명을 드릴 필요가 있는데요. 인자로서 true 를 지정하게 되면, 단순히 요소만을 복사하는 것이 아니라, 그 요소에 달려있는 이벤트 처리기(예, click, mouseover)등도 복사가 된다는 것입니다. 인자가 없는 기본 clone() 메서드로 복사하게 되면, 요소 자체만 복사될 뿐, 해당 요소에 달려있는 이벤트 처리기들은 복사가 되지 않습니다.

다음과 같은 htm 페이지를 하나 작성해 보도록 해요.
Manip.htm

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title></title>
  <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
  <script type="text/javascript">
    $(document).ready(
      function() {
        $(".annotation").each(function(i) {
            var idx = i + 1;
            $(this).after("<sup>" + idx + "</sup>");
        });

        $("button").click(function() {
            alert($(this).val());
            $(this).remove();
        });

        var $prev = $("#prevBtn").clone();
        var $next = $("#nextBtn").clone(true);

        $("div#bottomDiv").prepend($prev);
        $("div#bottomDiv").append($next);
    });
  </script>
  <style type="text/css">
    * { font-size:12px; font-family:돋움; }
    .annotation { background: pink; }
  </style>
</head>
<body>
  <button id="prevBtn">이전 페이지로</button>
  <button id="nextBtn">다음 페이지로</button>
  <h3>jQuery 란?</h3>
  <div>
    <b>jQuery</b>는 가볍고 빠르며, 간결한 <font class="annotation">오픈소스</font> 스크립트 라이브러리입니다. 이를 이용하면 Rich 웹 UI 를 개발하는 데 도움이 되는 다양한 기능들 즉, HTML 문서  <font class="annotation">트래버스</font>, 이벤트 처리, 애니메이션, <font class="annotation">Ajax</font> 상호 작용 등을 지원하여 빠르고 견고하게 리치 웹 애플리케이션 개발을 할 수 있도록 지원합니다.
  </div>
  <br />
  <div id="bottomDiv"></div>
</body>
</html>
```

jQuery 코드는 주석 표현을 화면에 삽입하는 것으로 시작하고 있습니다. append() 메서드를 설명할 때 예시했던 소스를 사용해서 말이죠. Annotation 이라는 css 클래스를 갖는 요소들을 루프 돌면서 <sup> 태그를 사용하여 주석을 표현하고

있는데요. 앞선 강좌 설명에서는 append 메서드를 사용했던 것에 반해, 이번 예제에서는 after 메서드를 사용해서 주석 표현을 하고 있습니다. 그 편이 좀 더 깔끔해 보여서요.
이어지는 코드는 <button> 요소들에 대해서 click 이벤트 처리기를 달고 있습니다. 그리고, click 처리기에서는 현재 버튼의 텍스트를 메시지박스로 나타낸 뒤, 그 버튼을 DOM 에서 제거하도록 remove() 메서드를 사용하고 있습니다.
그리고, 마지막 코드 구역은 이전 페이지 버튼과 다음 페이지 버튼을 복사해서(clone), 하단의 div 영역에 append 하는 것을 볼 수 있습니다. 다만, 이전 페이지 버튼은 clone() 메서드로 복사를 했고, 다음 페이지 버튼은 clone(true) 메서드로 복사를 했기에, 전자는 클릭 이벤트 처리기가 무시되며, 후자는 click 이벤트 처리기까지 포함된 상태로 복사가 된 것을 확인할 수 있을 것입니다. 즉, 복사된 이전 페이지 버튼은 클릭해도 아무런 반응이 없지만, 다음 페이지 버튼은 메시지박스가 뜬다는 것이죠.

5. jQuery :탐색(Traversing) 메서드

트래버스라는 단어의 의미는 어떤 것을 횡단하거나, 탐색하거나, 검토하거나, 여기 저기 왔다갔다 하는 등의 의미를 갖는 단어입니다. 즉, jQuery 의 트래버스 관련 메서드들은 개체들의 집합에서 다시금 특정 개체를 찾거나, 필터하거나, 추가하는 등의 작업을 위해 지원되는 API 메서드들의 그룹이라고 보시면 됩니다. 사실, 개체를 찾거나 필터하는 작업은 대부분 셀렉터를 통해서 이루어지지만, 1 차적으로 셀렉터를 통해서 필터링을 하고 난 다음에, 그 결과 집합 내에서 추가적으로 필터링하거나, 추가 탐색하거나, 다른 결과 집합과 합치거나 하는 등의 작업을 하기 위해서는 이러한 메서드가 유용하게 사용됩니다.
우선, 추가 필터링 관련 메서드들부터 알아보도록 하겠습니다. 다시 말씀드리지만, 이들은 모두 셀렉터를 통해서 1 차적으로 일치되는 집합을 얻어낸 후, 그 집합에 대해서 사용할 수 있는 메서드들입니다.

eq(index)	일치된 요소들 중에서 인덱스와 일치하는 단일 요소를 가져옵니다.
filter(expr)	지정된 표현식과 매치되지 않는 모든 요소들을 제거합니다. 즉, 매치되는 요소들만을 가져옵니다.
filter(func)	지정된 함수와 매치되지 않는 모든 요소들을 제거합니다.
is(expr)	현재 개체와 표현식에 해당한다면 true, 표현식에 여러 개의 조건이 있다면 그 중 한 개만 맞아도 true 가 됨
map(callback)	jQuery 개체 안에 있는 요소들의 집합을 다른 집합으로 변경해서 옮긴다.
not(expr)	지정된 표현식과 매치되는 모든 요소들을 제거합니다. 즉, 매치되지 않는 요소들만을 가져옵니다.

이 중 특히나, filter 메서드와 is, not 메서드는 활용빈도가 높습니다. 셀렉터에 의해서 1 차적으로 걸러낸 데이터를 가지고 작업을 하다가 2 차적으로 추가 필터링을 한다거나, 특정 요소가 어떤 조건에 부합하는지를 검사한다거나 하는 작업은 꽤나 자주 하게되는 일이니까요.
메서드의 인자인 expr 은 expression 을 의미하는 것으로 여기에는 주로 셀렉터를 사용합니다.
eq(index)라는 메서드는 사실 :eq(index) 라는 셀렉터 필터와 거의 동일합니다. 하나는 메서드이고 하나는 셀렉터 구문이라는 차이만 있는 것이죠. 다음 예제 코드를 한번 보세요.

```
$("div:eq(1)").addClass("blue");
$("div").eq(1).addClass("blue");
```

첫 번째 구문은 이미 익숙한 셀렉터이죠? 다들 아시겠지만, 현재 문서에 있는 div 중 2 번째로 등장하는 div 를 얻어내는 셀렉터입니다. 그리고, 그 요소에 대해 blue 라는 css 클래스를 적용하고 있습니다.
이는 eq() 메서드를 사용하는 두 번째 구문처럼 작성할 수도 있습니다.일단 모든 div 들을 셀렉트한 다음, 그 중 2 번째 개체를 얻어내기 위해서 eq(1)이라는 메서드를 사용하는 것이죠. 사실, 이 둘은 결과적으로 동일합니다. 그렇다면, 어떤 방법을 쓰던지는 우리의 맘일까요? 그럼요~ 여러분의 마음입니다. 다만!!!
eq() 메서드를 사용한 경우에는 end()라는 엄청난 초능력자 메서드의 도움을 받아 이전 상태로 돌아갈 수 있다는 장점이 있습니다(eq() 메서드 뿐만 아니라 대부분의 조작 메서드는 end() 메서드를 사용할 수 있습니다).

end() 메서드 :
현재 일치된 개체 집합을 변경하여, 방금 일어난 "파괴적인 작업" 직전의 상태로 되돌린다.
대부분의 조작 메서드, 트래버스 메서드에 end()라는 메서드를 적용할 수 있습니다. 이는 자기가 되돌릴 과거 상태가 존재한다면, 현재의 개체집합을 그 상태로 되돌리고, 되돌릴 과거없는 상태라면 단순히 빈 값을 반환합니다.

앞의 예제에 이를 사용해 보기 위해서, 다음과 같은 조건에 맞는 셀렉터를 한번 작성해 보도록 해요.

"모든 짝수 번째 div 를 구한 다음, 그 중 첫 번째 div 는 orange 라는 배경색을 적용하고, 두 번째 div 는 blue 배경색을 적용하라"
end() 메서드를 모르는 상황이었다면 여러분은 이를 다음과 같이 작성할 듯 합니다.

```
$("div:odd").eq(0).css("background", "orange");
$("div:odd").eq(1).css("background", "blue");
```

우선, $("div:odd")를 통해서 짝수번째 div 들을 구한 다음, eq() 메서드를 사용해서 그 중 첫 번째와 두 번째 요소에 각각 접근하는 것이죠. 코드가 두 줄이라는 것은 그렇다치더라도, 각각에 대해서 동일한 검색을 두번이나 반복하는 부분은 썩 마음에 들지 않습니다.
해서, 이 코드를 다음과 같이 할 수 있으면 좋을 것 같다는 생각이 들지 않나요?

```
$("div:odd")
        .eq(0).css("background", "orange")
        .eq(1).css("background", "blue");
```

그러면 참 좋겠습니다만, 안타깝게도 이는 올바로 동작하지 않습니다. 왜냐하면, 이미 $("div:odd").eq(0) 으로 인해서 매치되는 집합이 전체 div:odd 가 아니라 단일 요소로 줄어든 상황에서는 eq(1)을 해봐야 아무런 대상도 찾을 수 없기 때문이죠.
바로 여기서 end()가 엄청난 능력을 발휘하게 됩니다. 상기 코드는 다음과 같이 작성할 수 있습니다.

```
$("div:odd")
        .eq(0).css("background", "orange")
```

```
        .end()      // $("div:odd")와 동일
        .eq(1).css("background", "blue");
```

여기서 end()는 eq(0)을 하기 직전의 상태로 되돌리는 역할을 합니다. 고로, end()를 사용하면 단 한줄의 메서드 체인으로 원하는 작업을 마무리 할 수 있게 되는 것입니다
end()는 jQuery 에서 대단히 심도있게 사용되므로, 잘 기억을 해두시기 바랍니다.

그럼, 이제 필터 메서드에 대해서 알아보도록 하죠.
filter(expr)은 결과적으로는 일반 셀렉터를 통한 검색하는 것과 동일하다고 보시면 됩니다. 다만, 셀렉터는 문서상에 존재하는 개체들을 대상으로 검색한다면, 이는 메모리상에 존재하는 개체 집합을 대상으로 필터링을 한다는 부분에서 차이가 있습니다.

즉, 셀렉터에 의한 1 차적 검색 집합에 대해서 다시 추가적으로 필터링(걸러냄)을 하고 싶은 경우에 사용하는 API 메서드라는 것이죠. 인자로는 필터링을 위한 표현식을 제공할 수 있고요. 그 표현식과 매치되지 않는 모든 요소들을 개체 집합(메모리)에서 제거하여 필터링된 결과를 얻습니다.

반면, 유사한 API 로 Filter(func)도 있는데요. 이는 인자로 표현식이 아닌 함수를 사용하지요. 즉, 필터링(걸러냄)을 함수를 통해서 보다 세밀하게 제어하고 싶은 경우에 사용합니다. 필터링 조건이 일반적인 표현식으로 설명하기 어려운 경우에 별도의 함수로 작성을 해서 그 안에서 어떻게 필터를 수행할 것인지를 지정하는 것이죠.

가장 간단하게 예를 들면, 다음 2 개의 표현은 결과적으로는 동일하다고 볼 수 있습니다.
```
$("div:odd")
$("div").filter(":odd")
```
다만, 앞의 예제와 마찬가지로 filter 메서드를 사용한 경우는 end() 메서드의 도움을 얻어 이전 상태로 되돌릴 수가 있다는 장점이 있죠.

예를 들면, 다음과 같이 말입니다. 다음 코드는 앞선 예제에서 $("div:odd")를 $("div").filter(":odd")로 변경해 본 것입니다.
```
$("div").filter(":odd")
    .eq(0).css("background", "orange")
    .end()      // filter(":odd")
    .eq(1).css("background", "blue")
    .end()      // filter(":odd")
    .css("color", "red");    //이 코드는 odd 인 div 들의 폰트색상을 red 로 적용한다.
```
이는 셀렉터를 사용하여 전체 div 를 1 차적 선택한 다음에 filter() 메서드를 사용하여 그 중 짝수번째 인 것들만 다시 필터링을 하고요. 그 짝수번째 div 들 중에서 첫 번째 녀석(eq(0))은 오렌지 배경색으로 설정하고, 다시 짝수번째 div 들 중 두번째 녀석(eq(1))은 파란 배경색으로 설정하고, 마지막으로 모든 짝수번째 div 에 대해서 빨간 폰트를 지정하고 있습니다. end() 메서드는 한번만 가능한 것이 아니라 메서드 체인 내에서 계속적으로 사용하여 방금 전 상태로 계속 올라갈 수 있습니다.

또한, filter(expr) 메서드와 반대되는 메서드로 not(expr) 메서드도 있습니다. Filter(expr) 메서드가 현재의 집합에서 expr 표현식과 매치되는 것만을 가져온다면, not(expr) 메서드는 현재의 집합에서 expr 표현식과 매치되지 않는 것들만을 가져옵니다. 그렇기에, 다음 두 표현식은 동일한 개체 집합을 갖게 된다고 볼 수 있습니다.
```
$("div").filter(":odd")
$("div").not(":even")
```

또한, 중요한 메서드로 is(expr)도 있습니다. 이는 그 이름이 의미하듯이, 개체를 비교하여 true/false 를 알려주는 메서드인데요. 예를 들어, 다음의 코드를 한번 살펴보시죠.
```
var $myDiv = $("div").eq(5);
if ($myDiv.is("div"))
{
  $myDiv.css("border", "4px solid yellow");
}
```
이 코드는 우선 div 중 6 번째 요소를 가져와서 $myDiv 라는 변수로 참조하고 있습니다. 일단, 요 부분에서도 드릴 말씀이 하나 있는데요. 일반적으로 jQuery 에서는 jQuery 개체를 일시적으로 참조하는 변수명에는 $로 시작하는 변수명을 쓰도록 은근히 권유하고 있습니다. 그래야 일반 변수들과 jQuery 개체변수가 쉽게 구분되기 때문이지요. 물론, 꼭 그래야 하는 것은 아니지만 가독성이 좋아지니 가급적 그렇게 하시길 권해봅니다.
어쨌든, 위의 소스는 $myDiv 변수에 6 번째 div 를 참조한 뒤, is(expr) 메서드를 사용해서 그 개체가 "div" 개체인지를 비교하여, 만일 "div"가 맞다면, 그 div 에 대해서는 노란색의 두꺼운 테두리를 갖도록 지정하고 있지요.
expr 인자에는 일반적으로 "div"처럼 단일 구문을 작성하곤 하지만, ","를 구분자로 사용하여 여러 개의 표현식을 나열할 수도 있습니다. 다음과 같이 말이죠.
```
$myDiv.is(".orange, .blue, .lightblue")
```
이는 $myDiv 가 orange 나 blue, lightblue css 클래스 중 하나라도 가지고 있으면 true 가 됩니다. ^^
그리고, 그리 자주 사용되는 메서드는 아닙니다만 map(callback)라는 것이 있지요. 사실, 이는 생각보다 사용빈도가 적어서 아주 자세하게 설명하지는 않겠지만(자세한 설명은 공식 사이트에서 샘플 코드를 살펴보시면 쉽게 이해가 될 겁니다), jQuery 개체 안에 있는 요소들의 집합을 다른 집합으로 변경해서 옮긴다는 것은 기억해 두시기 바랍니다. 예를 들어, div 안에 각각 a 부터 z 까지 소문자 알파벳이 기록 되어 있는 상황에서 다음과 같은 코드를 사용한다면

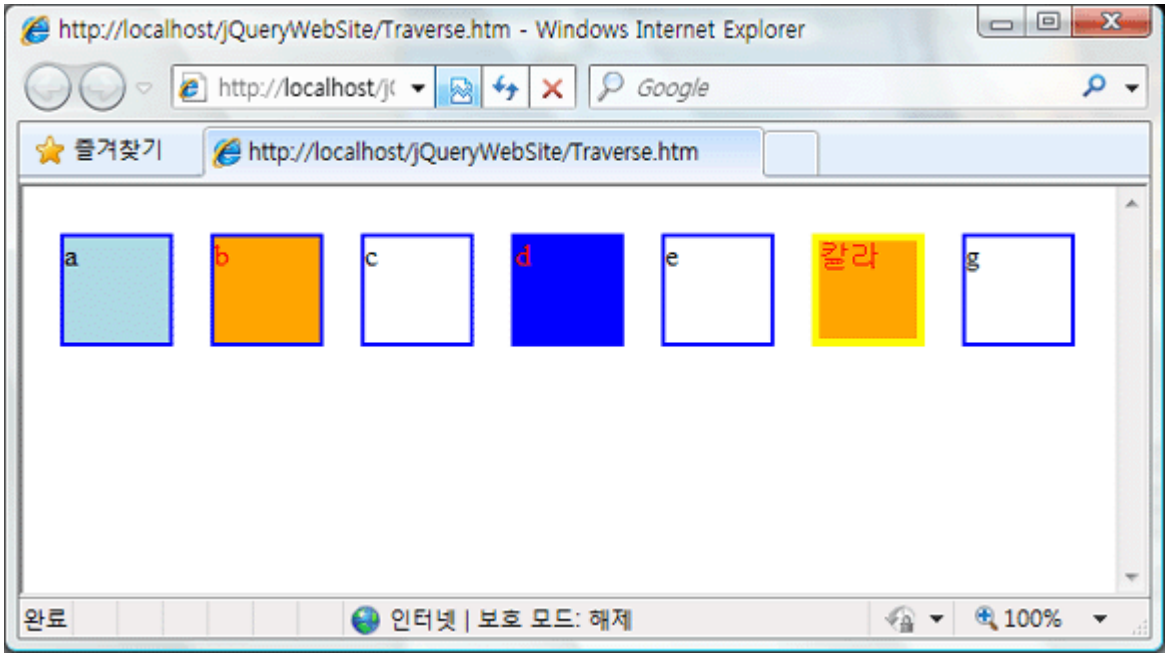
```
var arr = $("div").map(function() {
    return $(this).text().toUpperCase();
});
```

arr 이라는 변수에는 A 부터 Z 까지 대문자로 구성된 배열이 채워진다는 것이죠. 셀렉터로 선택된 개체들의 집합에서 원하는 데이터를 별도의 다른 집합으로 옮길 수 있다는 것이 이 메서드의 특징입니다. ^^

자. 그럼 지금까지 설명한 것이 모두 포함된 예제를 한번 같이 해볼까요? 다음은 Traverse.htm 페이지의 소스입니다.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title></title>
    <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
      <script>
          $(document).ready(function() {

              $("div").eq(0).addClass("lightblue");

              $("div").filter(":odd")
                .eq(0).css("background", "orange")
                .end()
                .eq(1).css("background", "blue")
                .end()
                .css("color", "red");

              var $myDiv = $("div").eq(5);
              if ($myDiv.is("div")) $myDiv.css("border", "4px solid yellow");
              if ($myDiv.is(".orange, .blue, .lightblue")) $myDiv.text("칼라");

              var arr = $("div").map(function() {
                  return $(this).text().toUpperCase();
              });
          });
    </script>
    <style>
      div { width:60px; height:60px; margin:10px; float:left;
            border:2px solid blue  ; }
      .blue { bac  kground:blue; }
      .lightblue { background:lightblue; }
      .orange { background:orange; }
    </style>
  </head>
<body>
  <div>a</div>
  <div>b</div>
  <div>c</div>
  <div>d</div>
  <div>e</div>
  <div class="orange">f</div>
  <div>g</div>
</body>
</html>
```

그리고, 다음은 이 페이지의 결과 화면이고요. 이미 앞서서 설명드린 코드들이 들어있는 것이기에 별도의 소스 설명은 필요없을 듯 합니다 ^^



이 부분을 꽤나 길게 설명드린 것은 그만큼 자주 사용하기 때문이니까요. 꼭 각 구문의 역할을 확실히 이해하시기를 부탁드립니다.

이번에는 트래버스 관련 메서드들 중 찾기와 관계된 API 들을 살펴볼까요.

find(expr)	지정된 표현식과 일치하는 요소를 검색한다.
add(expr)	현재 요소에 expr 와 일치하는 요소를 추가한다. 즉, 일치되는 요소가 추가 확장되는 의미이다.
next(expr)	현재 요소 바로 다음에 나오는 형제 요소를 선택한다. expr 을 지정하면 그로 필터링을 수행한다
nextAll(expr)	현재 요소 바로 다음에 나오는 모든 형제 요소들을 선택한다. expr 을 지정하면 그로 필터링을 수행한다
parent(expr)	현재 요소의 부모를 선택한다. expr 을 지정하면 그로 필터링을 수행한다
parents(expr)	현재 요소의 고유한 부모 요소들을 선택한다. expr 을 지정하면 그로 필터링을 수행한다
prev(expr)	현재 요소보다 앞서 나오는 형제 요소를 선택한다. expr 을 지정하면 그로 필터링을 수행한다
prevAll(expr)	현재 요소보다 앞서 나오는 모든 형제 요소들을 선택한다. expr 을 지정하면 그로 필터링을 수행한다
siblings(expr)	현재 요소의 모든 형제 요소들(자신은 제외)을 선택한다. expr 을 지정하면 그로 필터링을 수행한다

메서드 명칭이 직관적이어서 각각의 API 의 역할이 쉽게 이해가 될 듯 합니다.
가장 재미있는 메서드는 그 이름 때문에 오해하기 쉬운 add(expr)인데요. 이는 셀렉터에 의해 1 차적으로 검색된 결과 집합에 추가적인 검색 집합을 합치는(add) 역할을 합니다. 말로 하니 설명이 좀 모호한데요. 다음 코드를 한번 보시죠.

```
$("div").add("p").css("background", "yellow");
```

add 란 이름 때문에 얼핏 보면, 마치 div 요소 안에다가 p 요소를 생성하여 추가하는 것으로 느껴질 수 있는데요. 사실은 검색 결과가 합쳐지는 것입니다. 즉, $("div").add("p") 라는 것은 $("div, p")와 같다고 볼 수 있다는 것이죠. 해서, 다음과 같은 구문은

```
$("div")
        .css("border", "1px solid red")
        .add("p")
        .css("background", "yellow");
```

모든 div 에게는 붉은색 테두리를 주고, div 에 더하여 모든 p 요소들에게 노란 배경색을 주는 표현이 됩니다. 즉, 모든 div 와 p 는 노란 배경색을 갖지만, 붉은 테두리는 div 만 갖게 되는 것이죠. ㅎㅎ
그 다음으로 이야기할 만한 것은 find(expr) 인데요. 이는 현재 검색된 요소의 자식을 대상으로 추가적인 검색을 수행합니다. 가끔 어떤 분들은 filter()와 find()를 헛갈려 하시더군요. 둘의 차이는 사실 아주 간단합니다. filter()는 검색된 결과 집합에서 그 개체들에 대해서 다시금 필터링을 하는 것이고요. find()는 검색된 결과 집합에서 찾는 것이 아니라, 결과 집합 내의 각 요소의 "자식"에게서 찾는 겁니다. 그래도 어렵다면, 다음의 차이를 구분하실 수 있으면 되는데요.

```
$("div:odd").filter("p")
$("div:odd").find("p")
```

위의 코드 중 첫번째 구문인 filter 구문의 결과는 "없음" 입니다. 왜냐구요? 셀렉터에 의해서 짝수번째 div 들만 검색한 다음, 그 div 들 중에서 p 라는 태그를 갖는 놈들을 가져올라고 하니 있을리가 없죠. 모두 div 태그들만 있는 집합에서 뜬금없는 p 태그로의 필터링은 뭔가요? 당황스럽죠? ㅎㅎ. 해서, filter() 메서드에서 인자로 요소명을 주는 경우는 사실 거의 없죠.
반면, find()에 의한 결과는 아주 잘 나올 것입니다(물론, div 요소의 하위로 p 요소가 있다고 가정했을 경우에 말이죠). Div 들 중에 p 태그를 갖는 놈이 있다면, 그 p 요소들으 find() 명령에 의해서 수루룩 일치대상이 될 것입니다.
알고나니 쉽죠? 항상 모든 일이 그런 것 같습니다. 알기 전에는 대단한 것처럼 보이지만, 막상 알고나면 별 것 아니죠.
하지만, 알고 모름의 작은 차이가 가끔은 차등 대접의 원인이 되기도 합니다요. 그래서, 어릴때 부모님이 공부해라 공부해라 하시는 것인가 봅니다.

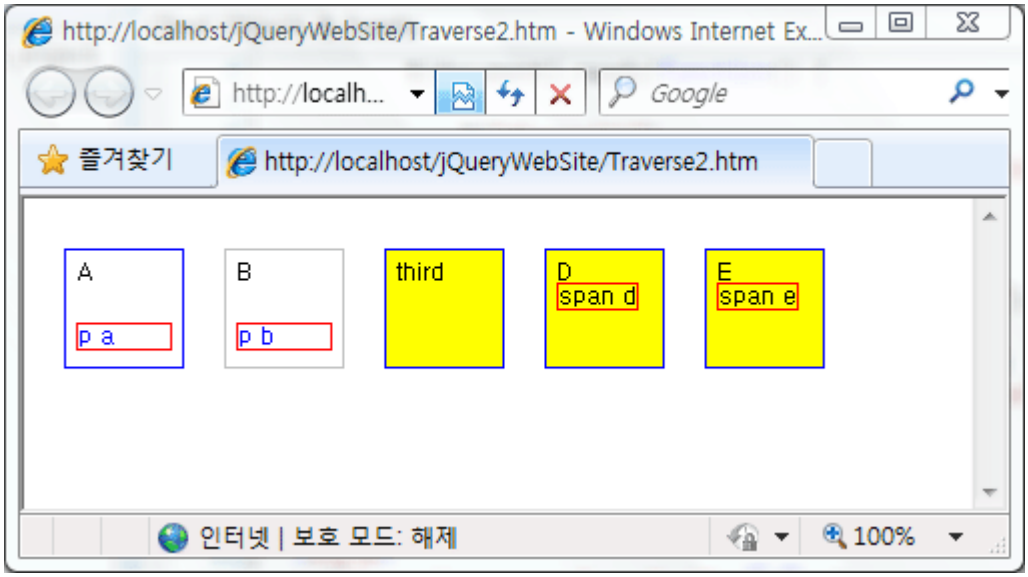
그 외의 메서드인 next(), prev(), nextAll() 이런 것들은 굳이 설명을 드릴 필요가 없을 것 같아요. 이름에서도 느껴질 뿐더러, 다른 언어에서도 이런 류의 메서드는 항상 있어왔고, 그 역할도 항상 똑같기 때문이죠 ^^
자. 방금 배운 것들을 그런대로 섞어본 종합 예제 하나 더 나갑니다.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title></title>
  <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
  <script type="text/javascript">
    $(document).ready(function() {

      $("div:eq(1)")
        .siblings().css("border", "1px solid blue")
        .end()
        .next().text("third")
        .end()
        .nextAll().css("background", "yellow");

      $("div").find("p").css("color", "blue")
        .add("span").css("border", "1px solid red");

    });
  </script>
  <style>
    * { font-size:12px; font-family:돋움; }
    div { width:60px; height:60px; margin:10px; float:left;
          border:1px solid silver; padding: 5px }
  </style>
</head>
<body>
  <div>A <p>p a</p> </div>
  <div>B <p>p b</p></div>
  <div>C <p>p c</p></div>
  <div>D <br /><span>span d</span></div>
  <div>E <br /><span>span e</span></div>
</body>
</html>
```

예제는 siblings()와 next(), nextAll() 그리고 find()와 add()까지 골고루 섞어서 사용하는 대단히 적절하다 못해 아니꼽기까지 한 출중한 예제가 아닐 수 없습니다.

```
$("div:eq(1)")
  .siblings().css("border", "1px solid blue")
  .end()     // $("div:eq(1)")
  .next().text("third")
  .end()     // $("div:eq(1)")
  .nextAll().css("background", "yellow");
```

라는 구문은 문서 상에 존재하는 div 중 두번째 div 를 1 차적으로 가져온 다음, 그와 친구들인 모든 요소들(siblings())에게 파란색 테두리를 적용하고요. end()를 사용해서 다시 두번째 div 로 돌아온 다음, 자신의 바로 뒤에 나오는 요소(next())의 텍스트를 third 로 바꾸고 있습니다. 그리고, 또 end()를 사용해서 두번째 div 로 되돌아 온다음, 자신의 뒤로 나오는 모든 형제 요소들에게 노란색 배경색을 지정하고 있습니다.
가급적 end() 메서드 뒤에는 // 주석을 사용하여, end()로 인한 대상 집합이 현재 무엇인지를 적어두시는 좋습니다. 그러면, 보다 코드가 직관적이 될테니까요 ^^
그러면, 브라우저로 이를 실행한 결과는 어떨까요? ㅎㅎ


http://localhost/jQueryWebSite/Traverse2.htm - Windows Internet Ex...
http://localh...
Google
즐겨찾기
http://localhost/jQueryWebSite/Traverse2.htm
A
p a
B
p b
third
D
span d
E
span e
인터넷 | 보호 모드: 해제
100%

6. jQuery : css 와 attr 메서드

CSS 관련된 메서드들과 어트리뷰트 관계된 메서드들도 설명은 하고 가야하지 않겠습니까? 이게 은근히 많이 사용되거든요~ 말이 길면 제 손가락만 피곤해 집니다. 바로 설명 들어가겠습니다.

CSS 관련 메서드

css(*name*)	매치되는 첫번째 요소의 스타일 속성을 반환한다. 예 : var color = $(this).css("color");
css(*name*, *value*)	매치된 모든 요소에 대해 단일 스타일 속성을 설정한다 예 : $(this).css('color':'yellow');
css(*properties*)	매치된 모든 요소들의 스타일 속성에 키/값을 설정한다. 예 : $(this).css({ 'color':'yellow','font-weight':'bolder'});

CSS Class 관련 메서드

addClass(*class*)	매치된 요소들의 각 집합에 지정된 CSS 클래스를 추가한다.
hasClass(*class*)	지정된 클래스가 매치된 요소 집합 중 최소 한 군데 이상 적용되어 있다면 true 를 반환한다.
removeClass(*class*)	매치된 요소들의 각 집합에서 지정된 CSS 클래스 혹은 모든 클래스를 제거한다.
toggleClass(*class*)	지정된 클래스가 적용되지 않았다면 적용하고, 이미 적용되어 있다면 제거한다.

이전 강좌들을 통해서, 수많은 메서드들로 단련된 여러분이라면 비록 머리 속에 식스팩은 없어도(응?), 이 정도의 메서드는 쉽게 이해가 가능하실 겁니다.

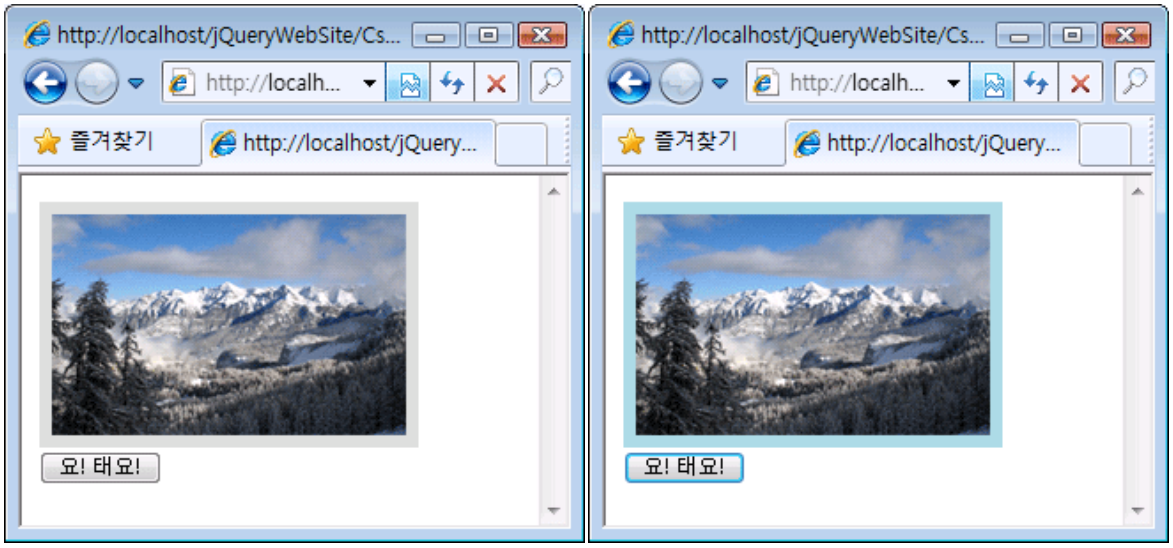
이들은 여러분이 정말로 자주 접하게 되는 CSS 속성을 직접 조작하거나, CSS Class 를 다루는 것과 관련된 메서드들이니까요. 굳이, 이 와중에도 포인트가 되는 부분을 찾으라면, JSON 형식의 구문을 사용하여 여러 CSS 속성을 한번에 설정할 수 있는 css(properties) 라는 오버로드 메서드가 존재한다는 것하고요. CSS Class 를 편하게 추가했다 제거했다 할 수 있게 하는 toggleClass()라는 메서드가 제공된다는 것이 포인트라 할 수 있습니다(적립되는 포인트는 아닙니다!! ? 허허.. 안타깝네요. 나이 40 을 바라보는 개그는 이런 것인가!)

뭐 워낙 설명이 주옥같아서 읽기만해도 이해가 되긴 하시겠지만, 그래도 예제는 한번 보고 가야겠죠? 자자~ 그렇다면 한번 씸쁠한 예제 하나 같이 해보죠.

CSS.htm

```
<html>
<head runat="server">
    <title></title>
    <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
      <script type="text/javascript">
          $(function() {
            $("#MtSnow")
               .addClass("winter")
               .css("width", "200");

            $("button").click(function() {
              $("#MtSnow")
                     .toggleClass("summer")
              });
          });
    </script>
  <style>
      * { font-size:12px; font-family:돋움; }
      .winter { border:7px solid #dddddd; }
      .summer { border:7px solid lightblue; }
  </style>
</head>
<body>

  <div id="Mt">
    <img id="MtSnow" src="resource/snowMt.png" />
  </div>
  <button>요! 태요!</button>

</body>
</html>
```

소스는 뭐 늘 그렇듯이 간단합니다.
일단 화면부터 보도록 하죠. 화면에는 이미지가 하나 존재하고요. 그 이미지의 id 는 MtSnow 입니다. 그리고, 버튼도 하나 있어요. 그리고, 약간의 CSS 클래스들도 존재하는데요. winter 라는 클래스와 summer 이라는 클래스가 그것입니다. 각 클래스의 역할은 각각 테두리를 약간 두껍게 하여 연한 회색이나 연한 파란색으로 나타나게 하는 것입니다.
jQuery 의 초기 함수에서는 MtSnow 란 아이디를 갖는 개체에게 winter 라는 CSS 클래스를 적용(addClass 메서드를 사용)하고 있는 것을 볼 수 있습니다. 그리고, 메서드 체인을 사용해서 개체의 너비(width)를 200 픽셀로 지정하고 있지요(css 메서드 사용).
그런 다음, 버튼에게 click 이벤트를 걸어두고 있는데요. Click 이벤트의 사용법에 대해서는 기존 강좌인 • "폼 필터 및 조작 메서드(each 등)" 강좌에서 설명을 드렸었습니다. 물론, jQuery 이벤트에 대한 좀 더 구체적인 설명은 다음 강좌에서 다룰 예정이니깐, 기존 강좌의 내용이 약간 부족하게 느껴지시더라도 양해해 주세요(부족하다고 하기엔, 기존 강좌에서 설명을 너무 잘해놓았었죠? 우핫핫!).
여하튼, click 이벤트에서는 $("#MtSnow").toggleClass("summer"); 라는 코드를 사용해서, MtSnow 란 개체에게 CSS 클래스가 토글되어 적용되도록 설정하고 있는 것을 볼 수 있습니다. 소스와 같이 toggleClass("summer") 메서드를 사용하면, 버튼이 클릭될 때마다, MtSnow 개체에게 summer 라는 CSS 클래스를 적용했다가, 제거했다가를 반복하게 됩니다. 즉, 첫번째 클릭 시에는 addClass("summer")를 하고, 두번째 클릭시에는 removeClass("summer")를 반복해서 수행한다는 것이죠. 이미 MtSnow 란 개체는 앞쪽의 코드로 인해 winter 라는 CSS 클래스를 가지고 있는 상황이기에, toggleClass("summer")를 적용하게 되면, 첫번째 클릭 시에는 개체가 winter 와 summer 클래스 모두를 갖게 되고, 다시 클릭하면 winter 만 갖게되는 방식으로 동작하게 될 것입니다. 충분히 결과가 예상이 되시죠? 그렇다면, 그 예상과 실제 결과가 동일한지를 확인해 보도록 하세요. 다음은 결과 입니다.



좌측은 처음 로드 되었을 경우의 화면이고요. 우측은 클릭을 했을 경우의 화면입니다. 그리고, 다시 또 클릭을 하면 좌측처럼 원래의 모습으로 돌아가겠죠? 그렇습니다!!!
매우 쉽죠? jQuery 의 CSS 관련 메서드들을 사용하면 이렇듯 쉽게 스타일을 변경하거나, CSS 클래스를 변경할 수 있습니다. 피곤하게 여러분이 관련 스크립트 코드를 반복해서 작성할 필요가 없어요~~~
좋습니다. 지금까지 CSS 관련 메서드들을 알아봤고요. 이번에는 알아두면 매우 유용한 또 하나의 기능인 어트리뷰트 관련 기능들에 대해서 알아볼께요.
기본적으로 HTML DOM 은 사용자 정의 어트리뷰트를 사용할 수 있도록 하고 있기에, jQuery 의 어트리뷰트 메서드들도 알아두면 특정 시나리오에서 매우 유용하게 사용할 수 있습니다. 물론, 사용자 정의 어트리뷰트의 사용에 대해서는 논쟁이 많은 편이긴 합니다. 사용하지 않는 것이 바람직하다는 의견과 적절하게 사용한다면 사용해도 무관하다가 팽팽하게 맞서 있긴 하지만, 개인적인 생각은 비록 혼란을 야기할 수 있는 기법이라 하더라도, 적절한 장소에 적절한 규칙을 가지고 사용한다면 나쁘지 않다는 쪽입니다. 마치, 복어가 비록 독을 가진 위험한 물고기이지만, 독만 잘 처리한다면 대단한 보양식이 되어주는 것과 같다고나 할까요? 헛? 관계 없는 이야기인가요? 그렇다면, 독초도 잘 쓰면 약이 된다는 비유는 어떨까요? 아…네… !! 네! 자꾸 뭔가에 비유할라고 애쓰지 말고 강좌에 집중하겠습니다.
우선, 어트리뷰트 관련 메서드들부터 살펴보도록 하죠.

Attribute 관련 메서드

attr(*name*)	메치된 첫 번째 요소의 특정 어트리뷰트에 접근하여 값을 가져온다. 만일, 지정된 어트리뷰트 명이 존재하지 않는다면 undefined 가 반환된다. 어트리뷰트에는 title, alt, src, href, width, style 와 같은 것들이 해당된다.
attr(*properties*)	모든 매치되는 요소들의 어트리뷰트를 키/값 개체로 설정한다.
attr(*key*, *value*)	모든 매치되는 요소들의 단일 어트리뷰트의 값을 지정한다.
attr(*key*, *fn*)	모든 매치되는 요소들의 단일 어트리뷰트에 대해 계산된 값을 지정한다.
removeAttr(*name*)	매치된 요소 각각으로부터 해당 어트리뷰트를 제거한다

사용방법은 위에서 살펴봤던 CSS() 메서드와 유사합니다. 단일 어트리뷰트 값을 읽어오거나, 설정할 수 있고요. JSON 포맷을 사용해서 여러 개의 어트리뷰트에 값을 줄 수도 있습니다. 다만, CSS() 메서드에서는 지원되지 않았던 독특한

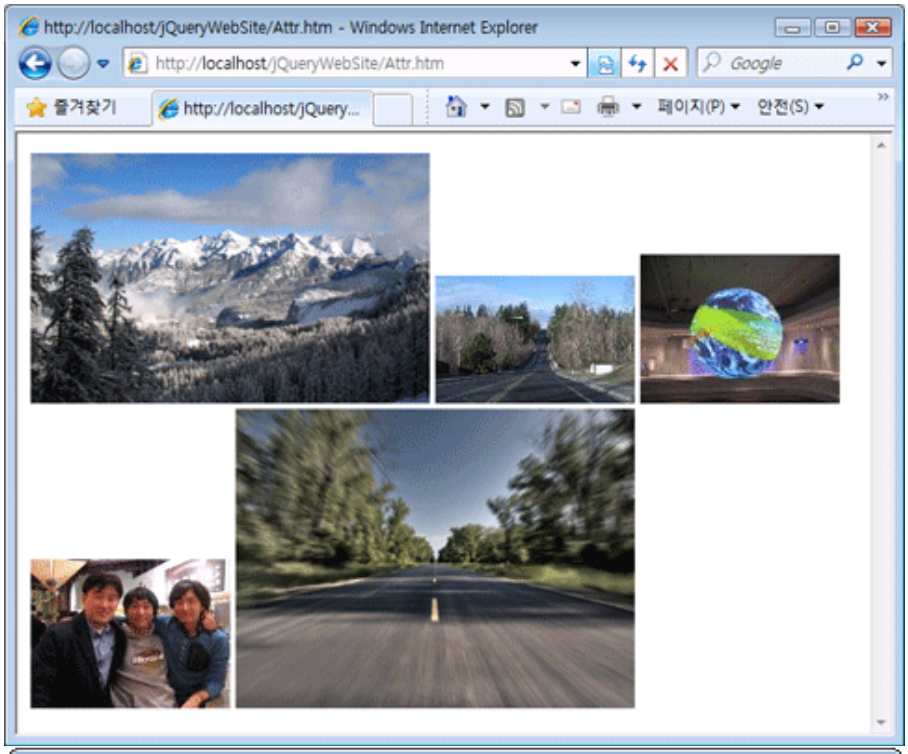
오버로드가 하나 있는데요. 그것은 두 번째 인자로 함수를 지정하여 그 함수의 계산된 반환 값으로 어트리뷰트의 값을 지정할 수 있는 기능입니다.
예를 들면, 다음의 코드를 한번 보시죠.

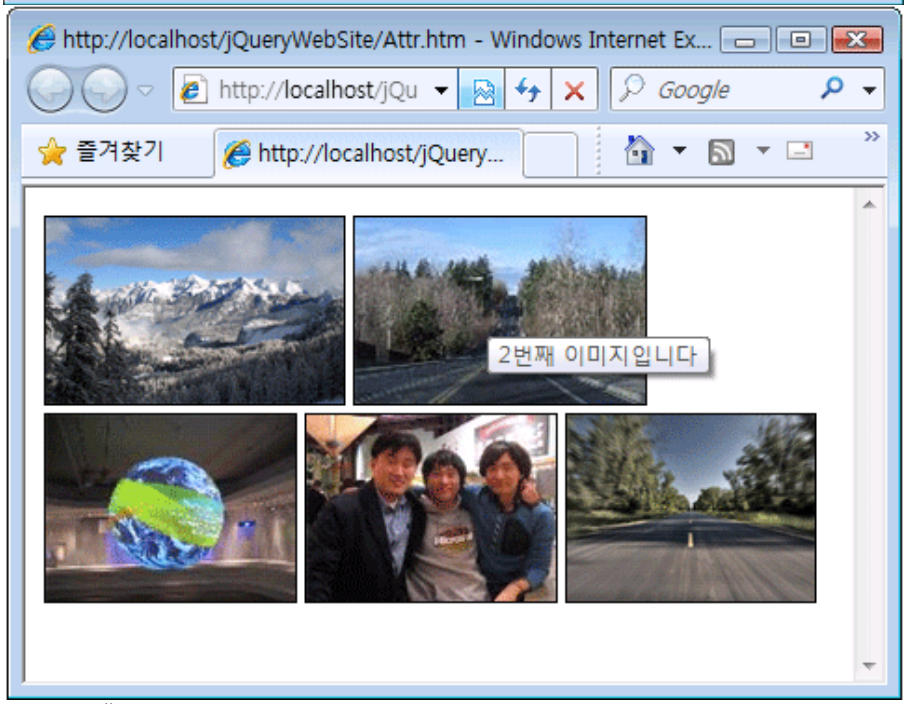
```
$("img").attr("title", function(i) {
                        return (i + 1) + "번째 이미지입니다";
          });
```

호오. 뭔가 재미있어 보이지 않나요?
위의 코드는 모든 <img> 들을 가져와서 그들의 title 어트리뷰트의 값을 동적인 문자열로 설정하는 예입니다. 두 번째 인자로 function 이 사용되고 있는 부분이 재미있는 부분입니다. 즉, 그 함수의 반환 값이 결과적으로 img 의 title 어트리뷰트 값이 되는 것입니다. function(i)의 인자인 i 는 0 부터 시작하는 요소의 인덱스 값이기에, 상기 결과에 의해서 각각의 이미지들은 "1 번째 이미지입니다", "2 번째 이미지입니다"와 같은 title 을 갖게 됩니다.
그렇다면, 예제로 이 부분을 한번 확인해 보도록 해요. 저는 다음과 같이 attr.htm 이라는 파일을 만들었습니다.
Attr.htm

```
<html>
<head runat="server">
    <title></title>
    <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
      <script type="text/javascript">
          $(function() {
              $("#photos img")
                .attr({
                     border: "1px",
                     height : "100"
                })
                .attr("title", function(i) {
                      return (i + 1) + "번째 이미지입니다";
                });
          });
    </script>
</head>
<body>

  <div id="photos">
    <img src="resource/snowMt.png" />
    <img src="resource/hongSeattle11.jpg" />
    <img src="resource/hongSeattle31.jpg" />
    <img src="resource/hongSeattle41.jpg" />
    <img src="resource/run.png" />
  </div>

</body>
</html>
```

코드는 그리 어렵지 않습니다. 화면에 놓여진 이미지들을 jQuery 의 attr() 메서드를 사용해서 정리하고 있는 것이 전부이니까요. 제가 사용하고 있는 각각의 이미지들은 width 와 height 가 서로 다르기 때문에, jQuery 로딩 함수에서는 각 이미지들의 어트리뷰트를 통일되게 설정하고 있는데요. border 은 1px 로, height 는 100px 로 고정하고 있는 것을 볼 수 있습니다. 그 다음, 조금 전에 설명드린 attr(name, function()) 오버로드를 사용해서 각 이미지에게 title 어트리뷰트 값을 동적으로 지정하고 있는 것을 볼 수 있습니다.
그렇기에, 이 jQuery 설정이 적용되고 나면, 최종 결과 화면은 이미지가 모두 1px 테두리를 가지며, 100px 의 높이를 갖는 이미지가 되고요. 마우스를 이미지에 올리면 각 이미지는 "2 번째 이미지입니다"와 같은 풍선 도움말이 뜨게 될 것입니다.
다음 그림은 jQuery 가 적용되기 전의 화면과 적용된 후의 화면을 각각 보여주고 있습니다.





사실, attr() 메서드가 이렇게 열을 낼 만큼 대단한 것은 아닙니다만, HTML DOM 개체에는 사용자 정의 어트리뷰트를 사용할 수 있기 때문에 약간 강조해서 설명 드리고 있습니다. 프로그래밍을 하다보면, 동적으로 생성된 특정 개체들을 식별해야 할 경우가 있으며, 그런 경우 사용자 정의 어트리뷰트를 사용하는 것은 꽤나 자주 있는 일입니다. 사용자 정의 어트리뷰트라는 것은 다음과 같이 임의의 어트리뷰트를 여러분이 설정하는 것을 말합니다.

```
<img src="myImage.gif" status="Big" order="desc" />
```

상기 태그에서 status 와 order 이라는 어트리뷰트가 바로 사용자 정의 어트리뷰트 되겠습니다. 그냥 특정 명칭을 마구 사용할 수 있습니다. 묻지도 따지지도 않습니다. 마치 '내가 그의 이름을 불러주기 전에는 그는 다만 하나의 몸짓에…'이라는 싯구처럼(또또! 이상한 비유든다!), 원래는 존재하는 어트리뷰트가 아니지만 지정되는 순간 살아 숨쉬기 시작하는 그런 동적인 어트리뷰트 기능입지요. 그렇기에, 난잡하게 사용할 경우 프로그래밍을 어렵게 한다는 단점이 있으며, 오타가 날 확률도 꽤나 있어서 이러한 사용자 정의 어트리뷰트의 사용은 그다지 권장되지 않는 편입니다. 하지만, 살다보면 어쩔 수 없이 사용할 수 밖에 없는 상황이 오곤 하죠. 그렇기에 그런 경우에는 부디 신중하게 사용할 것을 요구하곤 합니다.

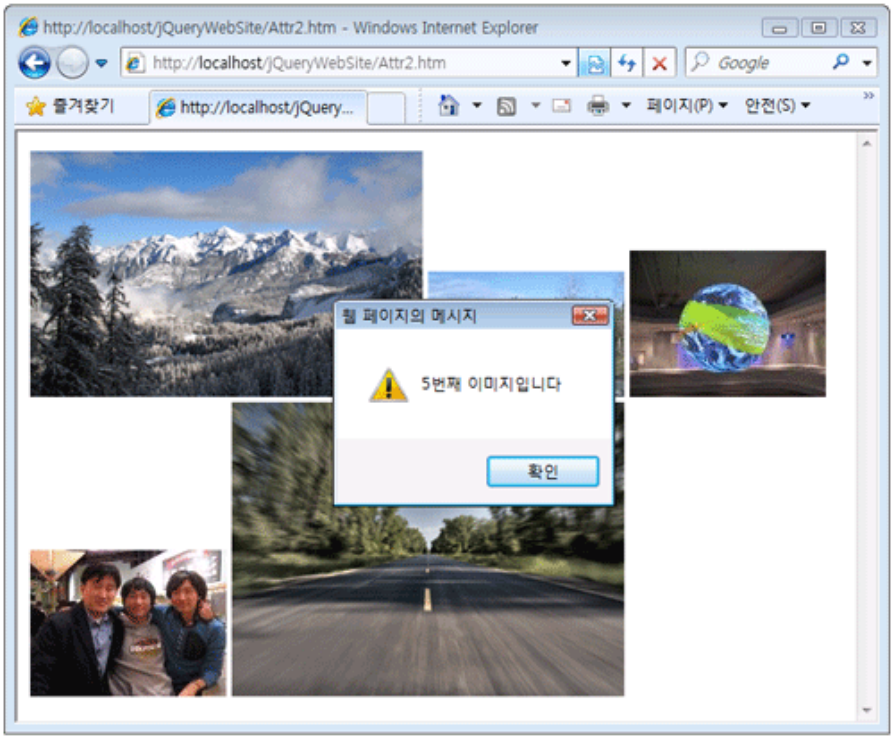
저의 경우는 주로 동적 넘버링을 위해서 사용자 정의 어트리뷰트 기능을 사용하곤 하는데요. 예를 들면, 각각의 이미지를 순서에 따라 식별하기 위해서 사용하곤 합니다. 한번 예를 들어볼께요. 이번 예는 각 이미지가 클릭되는 경우, 그 이미지가 몇번째 이미지인지를 메시지박스로 나타내는 예입니다.

Attr2.htm

```
<html>
<head runat="server">
    <title></title>
    <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
```

```
        <script type="text/javascript">
            $(function() {
                $("#photos img")
                    .each(function(i) {
                        //this.idx = i + 1;
                        $(this).attr("idx", i + 1);
                    })
                    .click(function() {
                        alert(this.idx + "번째 이미지입니다");
                    });
            });
    </script>
</head>
<body>

  <div id="photos">
    <img src="resource/snowMt.png" />
    <img src="resource/hongSeattle11.jpg" />
    <img src="resource/hongSeattle31.jpg" />
    <img src="resource/hongSeattle41.jpg" />
    <img src="resource/run.png" />
  </div>

</body>
</html>
```

소스는 각 이미지를 루프 돌면서, 그 이미지에게 idx 라는 어트리뷰트를 지정하고 있습니다. 원래 img 에는 idx 라는 어트리뷰트가 존재하지 않지만, 지금과 같이 무턱대고 지정해버리면 그 순간 해당 어트리뷰트를 가지게 되는 것입니다. 일단, 특정 어트리뷰트가 만들어지고 나면, 어디에서나 그 어트리뷰트를 읽거나 변경할 수 있습니다. 해서, click 이벤트에서는 그 idx 값을 읽어서 메시지박스로 출력하도록 하고 있습니다.
주석에서 보이는 것처럼, this.idx = i + 1; 라는 코드를 사용해서도 마찬가지의 결과를 얻을 수 있습니다(결과적으로는 같은 코드이니까요). this.idx = i + 1;은 image DOM 개체의 idx 속성에 값을 할당하는 코드이지만, 해당 속성이 존재하지 않으면 그에 해당하는 DOM 어트리뷰트를 먼저 만들고 난 다음에 값을 할당하게 됩니다. 쿨하네요!



이러한 사용자 정의 어트리뷰트를 사용하면, 데이터베이스로부터 가져온 각각의 사진 설명을 image 의 description 어트리뷰트나 detailInfo 어트리뷰트(어떤 이름이던 여러분이 원하는대로 지정)를 사용해서 할당하고, 그 설명을 jQuery 를 사용해서 이미지에 오버레이 텍스트로 출력한다거나 하는 것을 생각해 볼 수도 있을 듯 합니다. 응용하는 것이야 여러분의 아이디어에 달린 것이니~~~ 뭐든지 가능하겠죠?
위치를 구하고 하는 부분은 플러그인 중에 유명한 Dimension 이라는 플러그인을 사용하는게 더 낫다고들 합니다

7. jQuery : Event 다루기

그렇습니다. 웹 프로그래밍에서의 이벤트란 사용자의 어떤 액션(개체를 클릭하거나, 마우스를 움직이는 행동)에 의해서 야기되는 사건을 의미하며, 그러한 이벤트에는 클릭, 마우스 오버, 키 입력 등이 있습니다. 그리고, 일반적으로 그러한 이벤트가 발생하는 경우 특정 로직을 수행하기 위해서 해당 이벤트와 특정 메서드를 연결해 놓곤 하는데요. 이러한 연결을 이벤트 처리기 매핑(event hanlder mapping)이라고 이야기하며, 매핑에 의해서 연결된 메서드들을 이벤트 처리기라고 합니다. Click, MouseOver, KeyPress 등등을 알고 계시죠? 그렇습니다. 이들이 이벤트 처리기들입니다.
예를 들면, 여러분은 자바스크립트를 사용해서 다음과 같이 이벤트 처리기를 매핑할 수 있습니다.
Window.onload = pageLoad;
이 코드에서 이벤트는 onload 이고요(관례적으로, 이벤트명 앞에는 on 을 써서 표기하곤 합니다). 이벤트 처리기는 pageLoad 입니다. 그리고, 그 둘을 연결하는 위의 코드를 이벤트 처리기 매핑 코드라고 부릅니다. 즉, load 이벤트가 발생하는 경우에는 pageLoad 라는 이벤트 처리기를 실행하라는 의미이죠. 코딩 좀 하는 분들은 이벤트 처리기(event handler)를 이벤트 핸들뤄~라고 원어 그대로 부르기도 합니다만, 어떻게 부르던 이해할 수만 있으면 됩니다(이렇게 말하고나니, 왠지 이벤트 핸들러라고 더 많이 부르는 듯한 느낌이…).
뭐. 다 아시는 이야기를 했는데요. 사실 이벤트라는 놈을 사용하는 것 자체는 하나도 복잡할 것이 없으며, 어려움없이 쉽게 사용할 수 있지만, 이벤트의 내부 동작방식을 이해하는 것은 살짝 어렵게 느껴질 수 있습니다. 만일, 여러분이 기존에 VB 나 C#으로 사용자 정의 이벤트 작성을 해봤다면 델리게이트 및 이벤트와 관련하여 다소 헛갈리는 피곤함을 느껴보신 경험들이 있으실 겁니다(책을 막 찾아보면서 어떻게든 이해하려고 애를 쓰곤 하죠). 여러분이 머리가 나빠서 이해가 잘 안되는게 아니고요. 첨에는 누구나 다 난해하게 느낍니다(아! 여러분은 한 개도 안 난해했다고요? 조나단 쉬웠다고요? 그럼, 저만 어려웠었나보네요!!! 흥!!).
여하튼, 이벤트란 놈은 그런 놈인데요. 지금부터 이야기할 jQuery 의 이벤트 관련 기능들은 대부분 이벤트 매핑을 쉽게 하는 방법에 대한 이야기입니다. 뭐 원래도 이벤트 매핑은 쉬운 편인데, jQuery 가 뭐 대단한 기능을 제공해 줄만한 게 있나?라고 생각했다가는 올 겨울에 썰매 좀 끌게 되실 겁니다(큰 코가 다쳐서 루돌프 코가 된다는 말 개그입니다).
기본적인 자바스크립트 이벤트 매핑 방식(위에서 보여드린 Window.onload = pageLoad;과 같은 것)도 나름대로 깔끔한 편이라 사용하기에 그리 큰 불편함은 없습니다만, 페이지의 구성이 복잡해지게 되면 사용상의 제약이나 코딩 상의 불편함이 생겨나곤 합니다. 예를 들어, 버튼 개체에 동일한 이벤트를 여러 번 추가해야 하는 경우를 생각해 봅시다. 즉, 페이지에 상단에서 button.onclick = button_click;와 같이 이벤트 매핑을 해 두었는데, 페이지의 하단에서 button.onclick = btn_click;와 같이 동일 이벤트에 대해 또 다른 이벤트 처리기를 덧붙여야 한다면 어떻게 해야 할까요?
그럴 일이 왜 있어! 말두 안돼! 그리고, 그런 일이 생기면 설계가 잘못된거야! 걍 코드를 전부 button_click() 함수 안에 다 넣었어야지! 라고 말씀하실 수 있습니다. 네네~ 물론, 일반적인 경우라면 그렇게 하면 됩니다. 하지만, 동적으로 페이지를 꾸미는 경우에는 그렇게 하기가 어려운 경우도 있습니다. 예를 들면, 마스터 페이지에서 클릭 이벤트를 매핑하고, 콘텐트 페이지에서도 이벤트 처리기를 다시금 매핑해야 하는 경우도 생각보다 자주 있을 수 있기 때문입니다(Load 이벤트의 경우는 꽤 자주 이러한 처리가 필요합니다).
또한, 어떤 개체에 동적으로 이벤트 처리기를 매핑해야 하는 경우도 있을 수 있습니다. 다만, 동적으로 이벤트를 매핑한다고 해서 기존 매핑을 뒤집어 써서는 안되겠죠.
이러한 다중 처리기 매핑의 경우는 기본 자바스크립트 매핑만으로는 설정하기가 다소 까다롭습니다. 심지어는 동일 이벤트에 대한 다중 매핑이 아예 불가능하다고 아시는 분들도 계시는데, 사실 불가능하진 않습니다. 이에 대해서는 유경상 수석의 ▪ "스크립트 가지고 놀기 (Java Script Tip : Interception)" 강좌를 한번 참고해 보시기 바랍니다. 하단의 예제에서 이러한 부분을 다루고 있습니다(호오~ 유물탱~ 역시 박식하심!).
그리고, 자바스크립트 기본 매핑 방식으로 설정하기가 좀 짜증이 나는 또 다른 예로는 "일회성 이벤트 매핑"을 들 수 있습니다. 일회성 이벤트 매핑이란 것은 특정 이벤트가 한번만 동작하고 그 이후로는 동작되지 않는 것을 말합니다. 예를 들면, 버튼이 처음 클릭되는 경우에는 처리기 메서드가 호출되지만 그 이후로는 클릭해도 메서드가 동작하지 않게 처리하는 것이죠. 기본 스크립트로 이러한 처리를 하려면, 버튼 클릭 이벤트 처리기 내부에서 이벤트 매핑을 제거하는 코드를 넣어야 하는데요. 사실, 이런 식으로 코딩하는 것은 뭔가 깔끔하지 않은 프로그래밍처럼 느껴지곤 합니다. 이벤트 매핑의 주체가 이벤트 처리기 자신이다보니 일종의 무한반복 호출 코드처럼 비춰질 수 있기 때문이죠.
이 외에도, 자바스크립트 기본 매핑을 사용할 경우 작업하기가 약간 복잡한 다양한 사례들이 존재합니다. 비록 작업이 불가능하지는 않지만, 뭔가 개발이 복잡해지고, 지저분해지며, 관리하기에도 난해해지게 된다는 것이죠. 그렇기에, jQuery 는 이벤트와 관련한 코드를 간결하게 작성할 수 있도록 돕는 다양한 기능 또한 제공하고 있습니다.
그리고, 그것이 이번 강좌의 주제이지요! (기승전결의 기를 너무 오래 끌었네요. 아~ 기가 빠지네요). 그렇다면, 봅시다.
jQuery 의 이벤트 지원 메서드들을!!!!!
jQuery 가 제공하는 이벤트 관련 메서드들

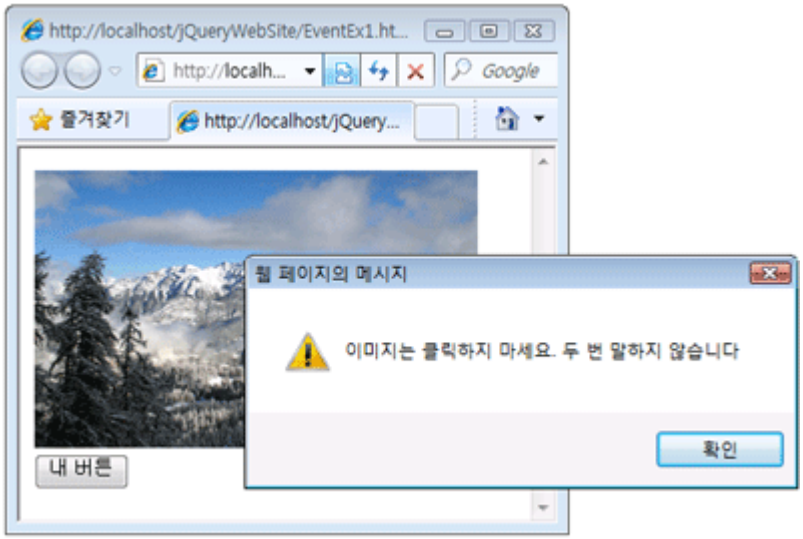
bind(type, data, fn)	매치된 요소에 이벤트 처리기를 바인딩한다. type 에는 이벤트 명칭을, data 에는 부가적으로 전달할 데이터 개체(Json)를, fn 에는 이벤트 처리기 함수를 작성하면 된다.
unbind(type, fn)	매치된 요소에서 지정된 이벤트와 매핑된 모든 처리기들을 제거한다. bind 메서드와 상반되는 메서드이다
one(type, data, fn)	매치된 요소에 오직 한번만 실행되는 이벤트 처리기를 바인딩한다
trigger(event, data)	매치된 요소의 특정 이벤트를 트리거한다.

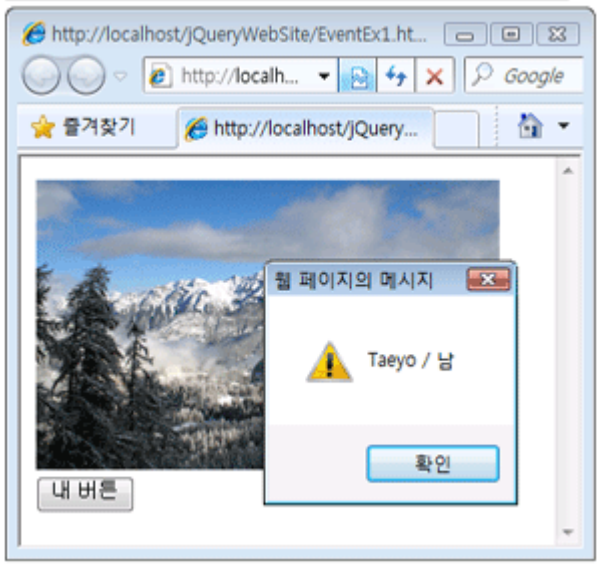
일관성을 중요시하는 jQuery 답게 역시나 메서드도 일관성이 있습니다. 사용하는 방법은 매우 간단합니다. 특정 개체에게 이벤트 매핑이 필요하다면 bind 메서드를 사용해서 해당 이벤트와 처리기를 등록하면 되고요. 이벤트 매핑을 제거하고 싶다면, unbind 를 사용해서 매핑을 제거할 이벤트를 지정하면 됩니다.
예를 들면, 다음의 코드는 MyBtn 이라는 아이디를 갖는 버튼에게 클릭 이벤트 처리기를 매핑하는 예입니다.

```
$("#MyBtn").bind("click", function() {
    alert("누가 내 버튼을 클릭한 것이냐?");
});
```
아! 물론, 이벤트 처리기를 익명 함수 대신 별도의 함수로 떼어내서 작성하는 것도 가능합니다만, 일반적으로 jQuery 에서는 익명 함수를 사용하곤 하죠. 사실, 실제로도 별도 함수로 분리하여 사용할 일은 적은 편입니다. 별도 함수로 작성하는 것은 대부분 재사용을 위해서인데, 이벤트 처리기는 재사용할 일이 거의 없기 때문이죠.
자. 이쯤 되면 물어오는 질문이 있습니다. 그렇습니다. 바로 첫 번째 인자로 사용하는 이벤트 명에는 어떤 것들이 있느냐는 것이 그것입니다. 하하하. 왜 안 물어보시나 했네요~ 그러면, 그러한 이벤트들을 정리해서 보여드리겠습니다. 짜잔!
사용 가능한 이벤트 명 : blur, focus, load, resize, scroll, unload, beforeunload, click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout, mouseenter, mouseleave, change, select, submit, keydown, keypress, keyup, error
그렇습니다. 여러분이 알고 있는 거의 모든 이벤트 명칭이 다 포함되어 있습니다.
반면, 이미 특정 이벤트에 매핑되어 있는 처리기를 제거하기 위해서는 unbind 메서드를 사용합니다. 사용 예는 다음과 같습니다.
```
$("#MyBtn").unbind("click");
```
이 코드를 사용하면, 이제 click 이벤트에 연결되어 있는 모든 처리기가 삭제되기에, 더 이상 클릭을 해도 아무런 동작도 하지 않게 됩니다. 간단하죠?
또한, jQuery 는 앞서 제가 언급했던 일회성 이벤트 처리를 지원하기 위한 one 이라는 메서드도 제공해 줍니다. 개인적으로 제가 참으로 좋아라하는 메서드입니다. 이는 이벤트 처리기를 딱 한번만 실행시키고 그 이상으로는 실행시키지 않는 일회성 매핑입니다. 다음 코드를 한번 보시죰.
```
$("#MyBtn").one("click", function() {
    alert("누가 내 버튼을 클릭한 것이냐?");");
});
```
이제, 클릭 이벤트는 처음 클릭하는 경우에만 메시지박스를 보여주고, 그 이후의 클릭은 아무런 반응도 하지 않을 것입니다. 멋지죠? 이 메서드는 생각보다 자주 사용되며, 꽤나 유용합니다. 꼭 기억해 두세요~
마지막으로, trigger()라는 메서드도 존재하는데요. 이는 특정 이벤트를 코드를 사용해서 일으키기 위한 메서드입니다. 즉, 실제로 사용자가 클릭 액션을 하지 않았음에도, 클릭을 한 것처럼 이벤트를 일으키는 코드라는 것입니다. 예를 들면, 다음과 같이 코드를 작성할 수 있는데요.
```
$("#MyBtn").trigger("click");
```
이 코드는 사용자가 버튼을 클릭한 것처럼 클릭 이벤트를 일으키게 합니다. 코드를 사용해서 강제로 이벤트를 발생시키는 방법이라는 것이죠. ^^
이로써, 4 가지 기본 이벤트 처리 메서드들을 알아보았는데요. 한가지 언급하지 않은 것은 각 메서드들이 두 번째 인자로 가지고 있는 data 라는 인자입니다. 혹시 그것이 무엇을 위한 것인지 궁금하시지는 않았나요? 아? 그런게 있는지도 몰랐다고요? 아…그..렇.. 군.. 요…
해서, 다시 한번 살펴보자면 bind 메서드의 기본적인 시그너처는 다음과 같았습니다.
bind(type, data, fn)
시그너처만 보면, bind 메서드에는 총 3 개의 인자가 제공되어야 하는데, 위에서 우리는 인자를 2 개만 작성했었죠? type 과 fn 을 말입니다. 허엇. 글고보니 인자를 2 개밖에 제공을 안했는데 어떻게 이것이 올바르게 동작되었던 것일까? 두번째 인자는 data 이지 fn 이 아닌데 말이죠.
두 번째 인자인 data 는 사실 선택적인 인자입니다. 즉, bind 메서드는 총 인자가 2 개가 들어오면 내부적으로 첫 번째 인자를 type 으로 인식하고, 두 번째 인자는 fn 으로 인식하게 됩니다. 그리고, 총 3 개의 인자가 지정되는 경우에만 첫 번째 인자를 type 으로, 두 번째 인자를 data 로, 그리고 세 번째 인자를 fn 으로 인식하는 것이죠. 해서, 앞의 예제에서는 인자를 2 개만 전달해도 잘 동작했던 것입니다. 하하. 이해가 되시죠?
사실, 두 번째 인자는 그다지 자주 사용되지 않는 편인데요. 이는 이벤트 처리기에 “부가적으로” 전달할 데이터 개체(Json 등)를 지정하기 위한 인자입니다. 그리고, 이 데이터 개체는 처리기 함수 안에서 [event.data.속성명] 과 같은 코드를 통해서 그 값을 읽어올 수가 있습니다. 말로는 설명이 좀 거시기 하네요. 예를 한번 볼까요? 그럽시다.
```
$("#MyBtn").bind("click",
                 { name: "Taeyo", gender: "남" },
                 function(e) {
                     alert(e.data.name + " / " + e.data.gender);
             });
```
보이다시피, 두 번째 인자로 { name: "Taeyo", gender: "남" } 라는 JSON 개체를 넘겨주고 있습니다. 이렇게 부가적인 데이터를 넘기면, 그 데이터를 이벤트 처리기 안에서 사용할 수가 있게 되는데요. 단, 그 데이터에 접근하려면 처리기 함수는 이벤트 데이터를 가져오기 위한 인자(예제의 경우, e)를 지정해 주어야 합니다. 그러면, e.data 라는 개체를 통해서 우리가 지정한 개체에 접근할 수가 있게 되는 것이죠. 예제에서 보이듯이 e.data.name, e.data.gender 와 같이 우리가 지정한 JSON 개체 속성에 접근하는 것을 볼 수 있습니다. 호옹~~ 뭔가 그럴싸하죠~ 그럴싸 하지만, 그다지 자주 사용하지는 않는다는 거~~ 그냥 기억만 해 두세요~
자자~~ 이로써 jQuery 의 기본 이벤트 지원에 대해서 알아보았습니다. 그러니 이 내용을 종합적으로 다루는 예제를 하나 해봐야 하겠죠? 안 그럼 뭔가 좀 서운하잖아요. 날도 추워지는데!
EventEx1.htm
```
<html>
<head runat="server">
    <title></title>
    <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
      <script type="text/javascript">
          $(function() {
```

```
            $("#MyBtn").bind("click",
                                { name: "Taeyo", gender: "남" },
                                function(e) {
                                    alert(e.data.name + " / " + e.data.gender);
                                });


            $("#MtSnow").one("click", function() {
                alert("이미지는 클릭하지 마세요. 두 번 말하지 않습니다");
            });
        });
    </script>
</head>
<body>

  <div id="Mt">
    <img id="MtSnow" src="resource/snowMt.png" />
  </div>
  <input type="button" id="MyBtn" value="내 버튼" />

</body>
</html>
```

소스에서는 앞서 설명했던 메서드들 중 bind 메서드를 사용하고 있으며, 제가 아끼는 애메서드(훗! 명칭이 므흣하네)인 one 을 실습해 보고 있습니다. 버튼의 클릭 이벤트는 앞서 설명했던 코드와 동일하기에 굳이 재설명할 필요가 없어보이고요. 이미지의 클릭 이벤트는 나름 개그로 한껏 멋을 부려 보았습니다. 개그 포인트는 “두번 말하지 않습니다” 이죠!!! 저 클릭 이벤트는 오직 한번만 실행될테니까요. 우핫핫!! 소스가 어떻게 동작할 것인지는 충분히 예상될 것 같은데요. 그 예상을 머리속에 그리시면서 결과 화면을 보시죠!!!





각각의 결과 화면은 이미지를 클릭했을 때와 버튼을 클릭했을 때의 결과입니다. 차이가 있다면, 버튼은 클릭할 때마다 메시지박스가 뜨는 반면, 이미지는 처음 클릭한 경우에만 메시지박스가 뜨고 그 이후에는 이미지를 아무리 클릭해도 반응이 없다는 것입니다. 바로 one 의 위대함입니다. 이벤트를 바인드할 수 있다는 사실은 확실히 이해하셨을 겁니다.

하지만!!

분명히 속으로 이렇게 생각하고 계시죠? **불편해!!!!** 라고!!

그렇습니다. bind 가 뭡니까! bind 가! 뭔가 좀 직관적이지 않다는 생각이 들 겁니다. 딴 언어에서는 클릭 이벤트는 click 이라고 쓰면 되는데, jQuery 는 뭔 bind("click"..) 이딴 식이야!! 이건 좀 별론데! 불편해!! 라고 생각하고 계시죠? 솔직히 말씀해 보세요~

그렇습니다. 불편합니다. jQuery 개발자들도 그것을 설마 모르겠습니까? 해서, jQuery 는 여러분이 좀 더 직관적으로 이벤트 매핑을 할 수 있도록, 자주 사용하는 이벤트들에 대해 각각 특별한 이벤트 도우미를 제공하고 있습니다. 전체 목록은 http://docs.jquery.com/Events 의 하단에서 확인하시기 바랍니다. 확인하기가 귀찮다면, 그냥 여러분이 애용하는 대부분의 이벤트에 대해 도우미가 제공된다고 생각하셔도 됩니다. 예를 들면, click, mouseover, focus 등등 여러분이 익히 알고 있는 이벤트 명칭과 동일한 메서드가 제공된다는 것입니다. 다만, 특이하게 dblclick 이벤트라는 것도 추가적으로 제공된다는 것은 기억할 필요가 있습니다. 그렇기에, 예를 들어보자면 앞선 예제의 이미지 클릭 이벤트는 다음과 같이 작성하실 수도 있습니다.

```
$("#MtSnow").click(function() {
    alert("...");
});
```

그리고, 이벤트를 트리거 하려면 trigger() 메서드를 사용할 필요없이 $("#MtSnow").click(); 이라고 호출하면 됩니다. 인자 없이 메서드를 호출하면 이것은 트리거의 역할을 하는 것이죠.

호오. 그렇습니다. 이런게 바로 우리가 원하는 방식입니다. 사실, 실무에서도 bind 메서드를 사용해서 이벤트 매핑을 지정하기 보다는 이러한 이벤트 도우미 메서드를 사용해서 매핑을 하는 편입니다. bind 메서드를 사용하는 것은 실제 코드에서는 거의 찾아보기가 힘들 정도입니다(잘 사용하지도 않는 것을 앞에서 왜 그리 열내서 설명한거야! 라고 불평하실 수도 있겠지만, 기초는 중요한겁니다!!!)

오홋홋! 다만, 이렇게 이벤트 도우미 메서드를 사용하는 경우에는 추가 데이터를 지정할 수가 없습니다. 즉, bind 메서드의 두번째 인자였던 data 인자(이벤트 메서드에게 추가적으로 전달할 데이터)를 사용할 수 없다는 것이죠. 하지만, 그렇다해도 큰 문제는 없습니다. 클로저(Closure)를 사용하면 이벤트 메서드 내부에서 바깥 쪽의 지역 변수에 접근할 수가 있으니까요. 클로저가 무엇인지를 지금 설명드리기에는 적절하지 않아 보이기에, 이에 대해서는 나중에 따로 설명을 드리도록 하겠습니다.

자. 그럼 몇몇 이벤트 도우미를 사용해보는 예제를 한번 다루어보도록 합시다.

```
<html>
<head runat="server">
    <title></title>
    <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
      <script type="text/javascript">
          $(function() {
              $("#Photo img").attr("src1", function() {
                  return this.src;
              })
              .mouseout(function() {
                  this.src = this.src1;
              })
              .mouseover(function() {
                  this.src = this.src2;
              })
              .dblclick(function() {
                  $(this).hide();
              });
          });
    </script>
</head>
<body>

  <div id="Photo">
    <img id="img1" src="resource/SEA12.png" src2="resource/SEA13.png" />
    <img id="Img2" src="resource/SEA13.png" src2="resource/SEA14.png" />
    <img id="Img3" src="resource/SEA14.png" src2="resource/SEA15.png" />
  </div>
</body>
</html>
```

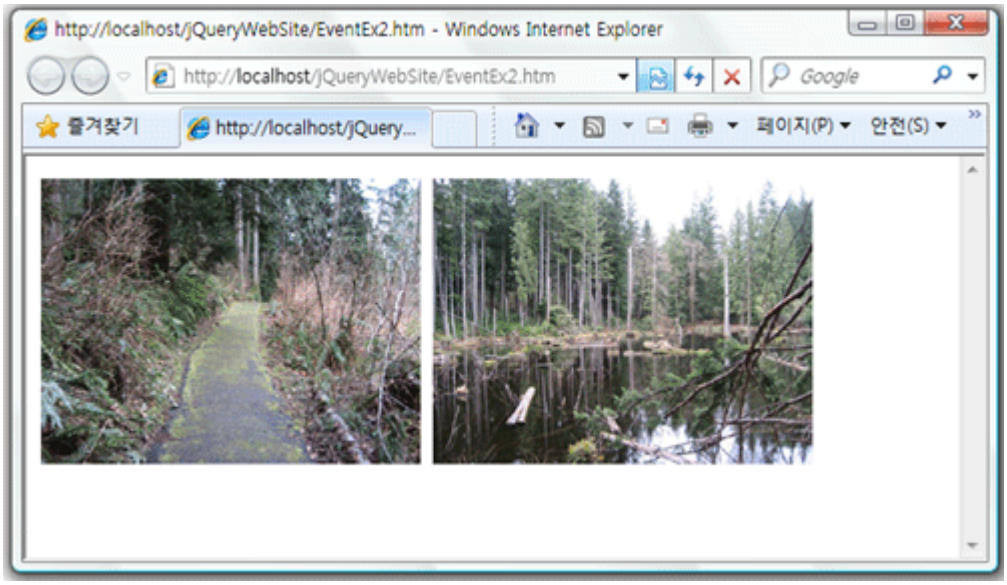
어려운 소스는 아닙니다만, 약간의 기교가 들어가있는 코드임을 눈치채실 수 있으시죠? 이게 뭐가 '기교'라는 거얏! 이라고 성질 부리지는 마십시오. 이 문장에서 중요한 단어는 “기교”가 아니라 고 앞에 있는 “약간의” 입니다. '약간의 기교' 우핫핫~~~

일단, 이 페이지는 여러 개의 이미지를 가지고 있는데요. 각 이미지에는 src2 라는 사용자 정의 어트리튜트가 지정되어 있는 것을 볼 수 있습니다. 사실 어트리뷰트의 명명 센스는 좀 구리긴 하지만, 어쨌든 그 어트리뷰트의 용도는 마우스가 이미지 위로 올라올 경우에 바꿔 보여지게 할 이미지 경로를 위한 것입니다.

자. 이제 jQuery 소스를 보도록 합시다. 소스에서는 이전 강좌에서 배운 attr 메서드를 사용해서 셀렉트된 각각의 이미지 요소에 대해 사용자 정의 어트리뷰트인 src1 을 지정하고 있습니다. 이는 원본 이미지의 경로를 저장해놓기 위한 것인데요. 만일, 이 코드를 사용하지 않는다면, 각각의 이미지 태그들은 원본 이미지 경로를 위해서 다음과 같이 작성되었어야 할 것입니다.

```
<img id="img1" src="resource/SEA12.png"
      src1="resource/SEA12.png" src2="resource/SEA13.png" />
```

강조하지만, 저는 사용자 정의 어트리뷰트의 사용을 권장하지는 않습니다. 이것은 어디까지나 예일 뿐이니까요.
설명이 빠르고 간결하면서도 친절하니깐 이해가 쉽고 확실하게 되시죠? 그러면, 계속해서 코드를 볼까요? 이어지는 코드는 3 가지의 이벤트 처리기를 지정하고 있습니다. mouseout, mouseover, dblclick 이 그것이죠. 또한, 이들을 멋지게도 메서드 체인을 사용해서 한방에 주욱 걸어주고 있습니다. 각각의 이벤트 처리기의 역할은 코드만 봐도 쉽게 이해가 될 정도로 간단합니다. mouseout 인 경우에는 해당 이미지가 원래의 이미지를 보여주도록 설정하고, mouseover 인 경우에는 대체 이미지(src2)를 보여주도록 설정하고 있습니다. 즉, 마우스가 이미지 위로 올라오고 나감에 따라 이미지가 바뀌어 보인다는 것입니다. 그리고, 따블클릭 시에는 해당 이미지를 눈에 보이지 않게 숨겨버리고 있죠. hide() 메서드는 일종의 애니메이션 메서드인데 이름만으로도 뭐하는 메서드인지 화끈하게 감이 오시겠지만 구체적인 설명은 추후 "애니메이션" 강좌에서 드리도록 하겠습니다. 지금은 대상 개체를 숨겨버리는 역할(즉, display:none)을 한다고만 알아두시면 되겠습니다.
이 얼마나 적절한 조화를 이루는 예제입니까? 이전 강좌를 복습까지 시켜주는 센스란!! 아아~
자. 그렇다면 결과 화면을 확인해 보도록 해요.



위의 결과 화면은 3 개의 이미지 중 하나를 더블클릭을 해서 사라지게 한 상태이고요. 오른쪽 이미지에 마우스를 올려서 원래의 이미지가 아닌 대체 이미지가 나오고 있는 상황입니다. 얼핏 보기엔 그냥 이미지 2 개를 올려둔 것처럼 보이시겠지만 그렇지 않아욧!
여러분도 직접 예제를 작성하고 해보시길 권해봅니다. 그러면, 예상대로 동작하는 것을 확인하실 수 있을 것입니다.
여기까지의 설명으로 jQuery 이벤트에 관한 70% 이상의 이야기를 드린 것 같네요. 남아있는 이벤트는 hover, toggle, live 등임.

8. jQuery 1.4 에대한소개 (상)

반드시 알아두어야 할 사항들을 중심으로 하여 전반적으로 내용을 훑어볼까 한다(그래도 꽤 내용이 많아서 2 회에 걸쳐 강좌를 올릴 예정이다).

1. 기본 기능

빠른 요소 생성

1.4 에서 새로 추가된 편리한 기능 중 눈에 띄는 첫 번째는 요소 생성을 보다 효율적으로 빠르게 할 수 있도록 생성 구문을 개선한 것이다. 예전에는 동적으로 단일 요소를 생성하려면 $(" ")와 같은 구문을 통해 요소를 생성한 다음, 어트리뷰트 지정하거나 해야 했지만, 이제는 추가 매개변수로 개체를 전달할 수가 있게 되었기에, 그 개체를 통해서 어트리뷰트와 이벤트들을 지정할 수 있다. 즉, 다음과 같은 구문을 사용하여 요소를 생성하면서 동시에 어트리뷰트 지정 및 이벤트 등록까지 한번에 할 수 있다는 것이다.

```
$("<div>",
{
    id: "foo",
    css: {
        height: "50px",
        width: "50px",
        color: "blue",
        backgroundColor: "#eeeeee"
    },
    click: function() {
        $(this).css("backgroundColor", "red");
    }
}).appendTo("body");
```

eq(-n)과 get(-n) 지원

1.4 부터는 eq()와 get() 메서드에 음수를 사용할 수 있다. 음수가 지정되면 그는 뒤에서부터 계산하여 개체를 반환하게 된다. 예를 들어, 다음의 두 구문은 각각 전체 td 들 중 뒤에서 두 번째 개체와 요소를 반환할 것이다.

```
$("div").eq(-2);
$("div").get(-2);
```

새로운 메서드: first(), last()

이전 버전에서는 필터로는 :first 와 :last 가 존재했지만, first()나 last()와 같은 메서드는 제공되지 않았었다. 1.4 에서는 이러한 메서드도 추가되었는데, 이 메서드들은 사실 내부적으로는 각각 .eq(0)과 .eq(-1)이다.

새로운 메서드: toArray()

jQuery 개체 집합에서 요소 배열을 얻어내기 위해서는 그 동안 get() 메서드를 사용해 왔을 것이다. 이제는 동일한 목적을 위해서 toArray() 메서드도 사용할 수 있다. 다만, get() 메서드는 인자로 숫자를 지정하여 배열 상의 특정 요소를 딱 집어서 가져올 수 있었지만, toArray() 메서드에는 인자를 지정할 수 없다. 이는 그 이름이 말해주듯이 오직 요소 배열을 얻어내기 위한 목적의 메서드이다.

jQuery() 메서드의 변경

$() 메서드 즉, jQuery() 메서드는 기존 1.3 버전까지만 해도 $(document)와 같은 의미였다. 하지만, 1.4 에 들어와서, 이는 빈 개체를 반환하도록 변경되었다. 이는 빈 개체를 만든 다음, 그 안에 동적으로 요소를 추가하고자 하는 경우 유용하게 사용될 수 있다. 반면, 이러한 변경으로 인해서 기존에 자주 사용하던 $().ready() 구문도 자제할 것을 권장하고 있다. 사실, 아직까지 그 구문은 호환성을 위해서 예외적으로 기존처럼 $(document).ready()로서 동작하도록 지원해주고 있긴 하다. 하지만, 더 이상은 이러한 구문을 권장하지 않으며, 확실하게 $(document).ready()나 $(function(){…}) 구문을 사용하는 것을 권장한다.

2. 어트리뷰트 관련 기능들

.attr() 메서드의 개선

attr() 메서드의 설정 값으로 함수를 사용할 수 있을 뿐만 아니라, 그 함수의 인자를 통해서 어트리뷰트의 현재 값을 알아낼 수 있게 되었다. 다음의 코드 예제를 보면 이해하기가 쉬울 것이다.

```
$("#photo")
    .attr("alt", function(index, value) {
        return "사랑하는 " + value
    });
```

만일, 소스에서 선택된 요소가 였다고 가정한다면, 상기 코드를 통해서 이미지의 alt 는 "사랑하는 우리 아가"로 변경될 것이다. 함수를 사용하여 값을 설정하는 것은 1.3 에서부터 이미 지원되었던 부분이지만, 함수 내부에서 인자(value)를 통해 alt 어트리뷰트의 현재 값을 알 수 있다는 부분은 1.4 에서 새롭게 개선된 부분이다.

.val() 메서드의 setter 함수 지원

attr() 메서드와는 달리 val() 메서드에 대해서는 기존에 setter 함수가 지원되지 않았다. 즉, val() 메서드의 인자로는 단일 값만을 지정할 수 있었을 뿐 함수를 사용할 수는 없었다는 이야기이다. 하지만, 1.4 부터는 여기에도 함수를 사용할 수 있으며, attr() 메서드와 마찬가지로 함수의 인자를 통해서 함수 내부에서 요소의 현재 값도 알아낼 수 있게 되었다.

setter 함수의 지원

방금 설명한 것과 같이, 1.4 부터는 attr() 및 val() 메서드에 대해서도 setter 함수가 지원되는데, 사실은 그 밖에도 수 많은 메서드들에 대해 setter 메서드가 지원되도록 확장되었다. 다음은 setter 함수가 지원되는 메서드들의 목록이다.

.css(), .attr(), .val(), .html(), .text(), .append(), .prepend(), .before(), .after(), .replaceWith(), .wrap(), .wrapInner(), .offset(), .addClass(), .removeClass(), .toggleClass()

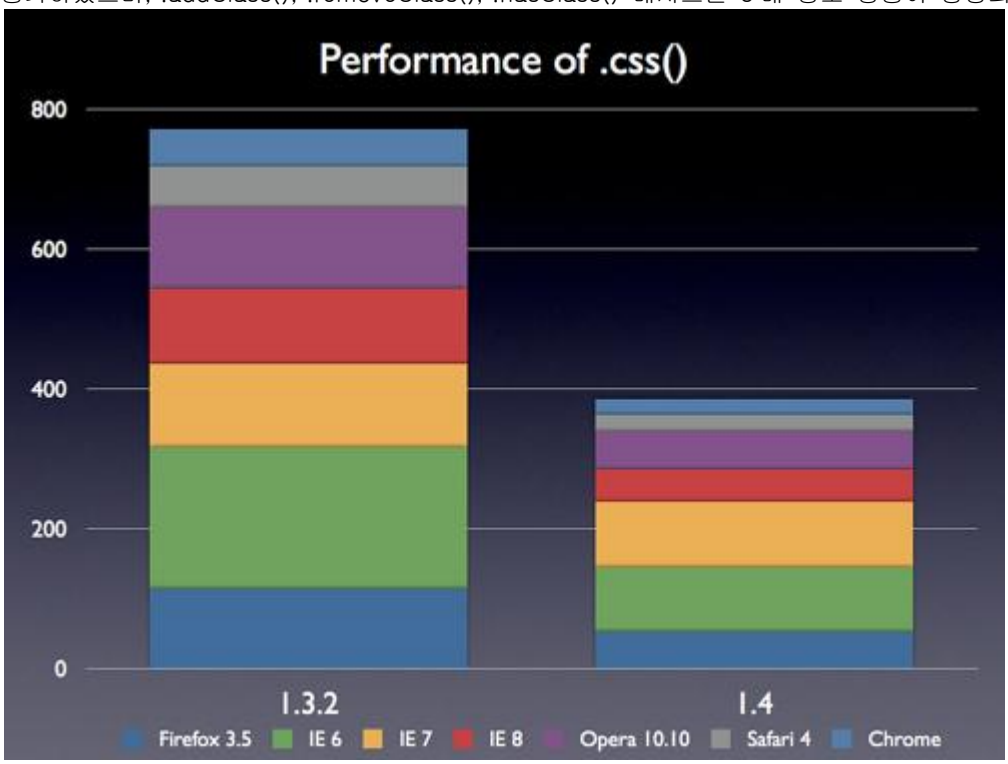
그리고, 다음은 이러한 메서드들 중에서도, attr()이나 val() 메서드처럼 함수의 인자로 요소의 현재값을 제공하는 메서드들을 다시 정리한 것이다.

.css(), .attr(), .val(), .html(), .text(), .append(), .prepend(), .offset(), .addClass(), .removeClass(), .toggleClass()

3. CSS 관련 기능들

성능 개선

가장 많이 사용되는 jQuery 메서드가 css 관련 메서드이다보니, 이 부분에 대한 성능적인 튜닝이 집중적으로 이루어졌으며, 그 결과, 1.4 버전에서는 이전과 비교하여 .css() 메서드의 성능이 2 배 가량
증가하였으며, .addClass(), .removeClass(), .hasClass() 메서드는 3 배 정도 성능이 향상되었다.



.toggleClass() 메서드 개선

기존 toggleClass() 메서드는 인자로 1 개의 css 클래스를 지정할 수 있었지만, 이제는 여러 개의 클래스를 지정할 수 있다. 여러 개의 클래스가 지정되면, 토글 시에 그 클래스들 모두를 합친 스타일이 적용될 것이다. 다음은 코드 예이다.

```
$("#myButton").click(function() {
    $("input").toggleClass("normal selected");
});
```

만일, 상기 클래스의 css 스타일이 다음과 같다면

```
.normal
{
     color:green;
}
.selected
{
     background-color:yellow;
}
```

이전의 toggleClass() 코드는 버튼이 한번 클릭되는 경우, normal 클래스와 selected 클래스가 함께 적용된 스타일(녹색 글자에 노란 배경색)이 적용될 것이며, 다시 클릭되는 경우 원래의 스타일로 되돌아 오게 될 것이다.

4. DATA 관련 기능

.data() 메서드 기능 개선

.data()라는 메서드는 모든 형식의 DOM 요소에 임의의 데이터를 저장하고 읽어올 수 있도록 돕는 유용한 메서드이다. 예를 들면, 다음과 같이 편하게 데이터를 저장하고 불러올 수 있었다.

```
$('body').data('foo', 52);
// boby 요소에 foo 라는 키로 52 라는 값을 저장한다
$('body').data('bar', { myType: 'test', count: 40 });
// boby 요소에 bar 라는 키로 개체를 저장한다

alert($('body').data('foo'));  // 52 를 메시지박스에 출력한다
alert($('body').data());    // 키를 지정하지 않으면 에러가 난다(1.3 의 경우)
```

다만, 1.3 버전까지는 데이터를 조회하는 경우, 키를 지정하지 않으면 에러가 발생하는 문제가 있었다. 1.4 에 들어서서는 이 부분이 개선되었는데, 즉, 키를 지정하지 않는 .data() 호출은 현재 해당 요소에 저장된 전체 개체를 반환하도록 변경되었다. 그렇기에, 1.4 를 사용한다면 상기 코드의 마지막 줄은 에러를 발생시키는 것이 아니라, 전체 저장값을 다음과 같은 json 개체로 반환하게 된다.

```
{foo: 52, bar: { myType: 'test', count: 40 }}
```

5. 탐색 관련 기능 추가

새로운 메서드: .has()

1.4 에서는 새롭게 :has() 필터와 동일한 역할을 하는 has() 메서드가 추가되었다. 메서드의 사용법은 일반적인 다른 탐색 메서드들과 동일하며, 인자로 셀렉터를 지정할 수 있다

새로운 메서드 : .nextUntil(), .prevUntil(), .parentsUntil()

새롭게 추가된 이들 메서드는 기존의 탐색 메서드인 .nextAll(), .prevAll(), .parents()와 많은 부분이유사하지만, 선택에 경계를 가질 수 있다는 점에서 약간은 다른 메서드이다. 즉, 기존의 .nextAll() 류의 메서드는 대상 요소 이후에 나오는 모든 형제 요소들을 선택하는 메서드였다면, 이번에 추가된 *Until() 메서드들은 대상 요소 이후 어디까지 선택해야 하는지를 지정할 수 있는 메서드라는 점이 차이이다. 말이 어렵게 느껴진다면, 예를 통해서 살펴보도록 하자.

만일, 다음과 같은 HTML 을 가지고 있다고 가정해 보자

```
<body>
    <p id="p1">p</p>
    <div id="div1">div</div>
    <p id="p2">p</p>
    <p id="p3">p</p>
    <div id="div2">div</div>
    <p id="p4">p</p>
    <div id="div3">div</div>
</body>
```

그리고, 다음과 같은 jQuery 구문을 사용한다고 해보자.

```
<script type="text/javascript">
    $("#div1").nextUntil("div").css("background-color", "red");
</script>
```

이는 div1 이라는 아이디를 갖는 요소에서 시작하여 그 다음 "div" 요소까지를 범위로 하여, 그 사이에 들어있는 모든 형제 요소를 선택하게 되기에, p2 와 p3 요소만이 빨간 배경색을 가지게 될 것이다.

만일, 상기 구문에서 nextUntil() 메서드가 아닌 nextAll()를 사용한다면, 그는 div1 이후에 나오는 모든 형제들 중에서 태그명이 div 인 것만을 선택하게 되기에, div2 와 div3 의 배경색이 빨갛게 될 것이다. 메서드의 차이가 분명하게 느껴지지 않는가?

재미있는 것은 nextUntil()에게 인자를 지정하지 않으면, 그는 끝점에 대한 경계가 주어지지 않은 것으로 인식되어 nextAll()과 완전히 동일하게 동작하게 한다는 것이다.

.add 메서드의 개선

이제 add() 메서드도 컨텍스트를 가질 수 있게 변경되었다. 사실, 일반적으로는 컨텍스트를 지정할 일이 거의 없지만, 특별한 상황에서는 컨텍스트가 필요할 수도 있다. 예를 들면, Ajax 요청에 대한 응답으로 어떤 개체를 받아온 경우, 동적으로 그 개체에게 add() 메서드를 사용해서 요소를 추가해야 한다면, 컨텍스트가 유용할 수 있다.

.closest() 메서드의 개선

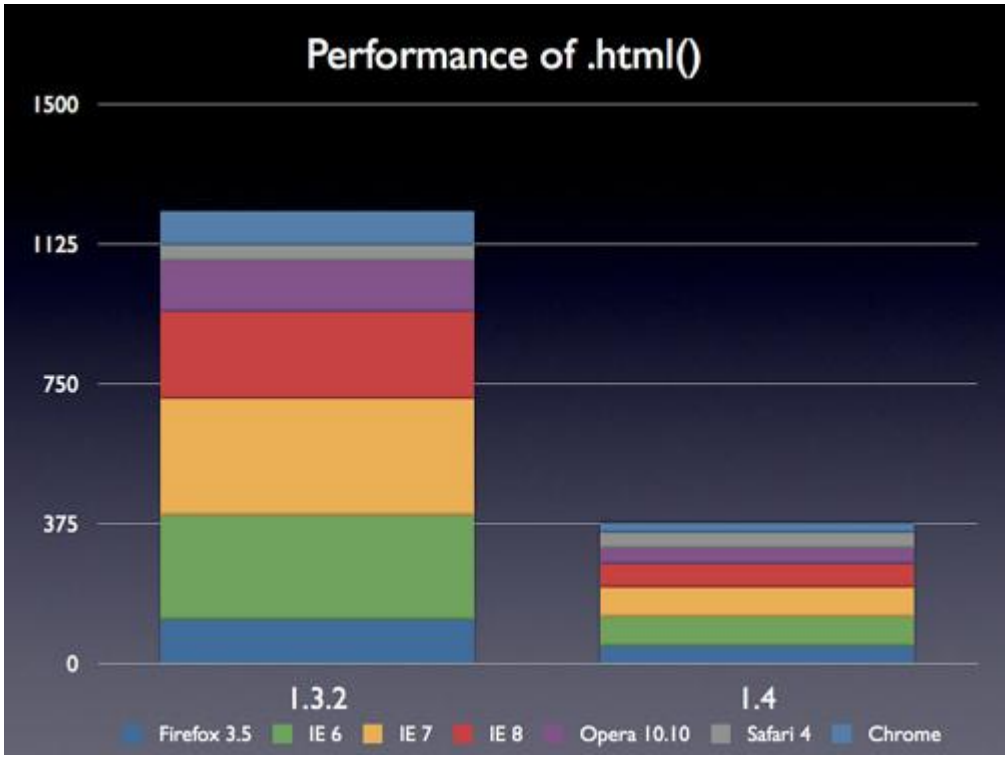
이제 closest() 메서드도 컨텍스트를 가질 수 있게 변경되었다. 선택된 요소와 가장 가까운 요소를 찾기 위한 메서드인 closest()에 컨텍스트를 지정한다면, 더욱 빠르게 원하는 요소를 찾아낼 수 있다.

9. jQuery 1.4 에대한소개 (하)

6. 조작 관련 기능 추가

성능개선

.append(), .prepend(), .before(), .after()와 같이 DOM 에 요소를 삽입하기 위한 메서드들의 성능이 전반적으로 향상되었으며 특히, html() 메서드의 경우는 기존과 비교했을 때, 3 배 이상 성능이 향상되었다. 다음은 jQuery 공식 사이트에서 제공하는 html() 메서드의 브라우저 별 성능개선 그림이다.



출처 : http://jquery14.com/day-01

뿐만 아니라, DOM 조작을 위해 자주 사용되는 .remove()와 .empty() 메서드의 경우는 4 배 이상 속도가 빨라지도록 개선되었다.

새로운 메서드 : detach()

동적인 조작을 돕기 위해서 detach() 라는 새로운 메서드가 추가되었다. 이는 DOM 에서 특정 요소를 제거하는 역할을 한다는 부분에서는 remove()와 기능적으로 동일하지만, 해당 요소의 데이터(이벤트 등)를 제거하지 않는다는 주요한 차이가 있다. 그렇기에, 이는 일시적으로 요소를 DOM 에서 제거했다가 다시금 DOM 에 추가해야 하는 경우 매우 유용하다. 이해가 잘 되지 않는다면 다음의 예제 코드를 살펴보도록 하자.

```
<script type="text/javascript">
    $(document).ready(function() {
        var $p = $("p").click(function() {
            $(this).css("background-color", "yellow");
        });

        $("p").detach();

        $p.css("color", "red");
        $p.appendTo("body");
    });
</script>
```

이 코드는 현재 본문에 Hello 라는 텍스트를 갖는 요소가 존재한다고 가정하고 작성한 코드이다. 코드는 우선, 그 P 요소에 클릭 이벤트 처리기를 지정하고 있다. 그리고, 그 요소를 $p 라는 변수를 통해서 참조하고 있는 것을 볼 수 있다. 그 다음, detach() 메서드를 사용하여 해당 요소를 DOM 에서 제거한다. 이 순간, 실제로 DOM 에서 P 요소는 제거된다(물론, 아직 $p 변수를 통해서 해당 개체는 참조되고 있다). 그 다음, 메모리 상의 참조인 $p 를 이용하여 폰트 색상을 변경하고 있으며, appendTo() 메서드를 사용하여 다시금 DOM 에 삽입하는 것을 볼 수 있다.

화면이 로드된 다음, p 요소를 클릭하면 배경색이 노랗게 바뀌는 것을 볼 수 있을 것이다. detach() 메서드를 사용해서 제거하였기에, 이벤트 정보가 유지되었기 때문이다. 반면, 상기 코드에서 detach() 메서드 대신 remove()를 사용한다면, 예제는 겉 보기에는 동일하게 동작하지만, 클릭을 해도 아무런 반응이 없을 것이다. remove()는 DOM 에서 제거할 뿐만 아니라 요소와 관계된 모든 데이터를 제거하기 때문이다.

before, after, replaceWith 개체의 기능 개선

이제 여러분은 DOM 에 아직 삽입되지 않은 노드(요소)에 대해서도 before, after, replaceWith 메서드를 사용할 수 있다. 이 이야기는 개체를 동적으로 생성한 뒤 DOM 에 삽입하지 않아도 메모리 상에서 조작을 할 수 있다는 의미이다.

```
$("<span>").before("<p>taeyo</p>").appendTo("div#box");
```

상기 코드는 span 을 동적으로 생성한 뒤, 그 앞에 taeyo 를 동적으로 만들어 붙이고, 그 전체를 box 라는 id 를 갖는 div 에 추가하는 예제이다. 이 코드를 실행시킨 뒤, $("div#box")의 html 을 확인해 보면 다음과 같은 결과가 나오게 된다.

```
<P>taeyo</P><SPAN></SPAN>
```

clone(true) 메서드의 개선

기존 clone(true) 메서드는 깊은 복사를 수행하긴 했지만, 개체에 연결된 이벤트까지 복사하지는 않았다. 1.4 부터는 이 부분이 개선되어, clone(true)를 수행할 경우 이벤트 및 data 까지도 모두 복사된다.

7. 이벤트 관련 기능들

이벤트 멀티 바인딩

1.4 부터는 이벤트 관련 바인딩을 개체로 구성하여 bind() 메서드에 지정할 수가 있다. 예를 들면, 다음과 같이 코드를 작성하는 것이 가능하다.

```
$("#box").bind({
    click: function() {
        $(this).text("click");
    },
    mouseenter: function() {
        $(this).text("mouseenter");
    },
    mouseleave: function() {
        $(this).text("mouseleave");
    }
});
```

상기 코드는 이벤트 바인딩을 익명 개체로 구성하여 바인딩하고 있지만, 해당 개체를 별도의 변수에 할당하여 전역적으로 선언한다면, 다른 곳에서 재 사용도 가능하다.

새로운 이벤트 : focusin, focusout

새롭게 지원되는 focusin, focusout 이벤트는 기능적으로는 focus, blur 이벤트와 완전히 동일하지만, 버블링을 지원한다는 것이 특징이다. 그렇기에, 여러분이 이벤트 위임 기능을 필요로 하는 경우에는 이러한 이벤트들이 큰 도움이 될 것이다. foucs 나 blur 이벤트는 live()를 사용하는 경우 제대로 동작하지 않을 것이니 말이다.

이는 focus 와 blur 는 버블될 수 없다고 정의해놓은 DOM 이벤트 명세에 따른 것이기에, 이벤트 위임이 필요하다면 새롭게 지원되는 focusin, focusout 이벤트를 사용하는 것이 바람직하다.

모든 이벤트는 live()와 함께 사용할 수 있다

1.4 부터는 거의 모든 이벤트를 live()와 함께 사용할 수 있지만, ready, focus, blur 이벤트는 예외적으로 여전히 사용이 불가능하다. 다만, focus 와 blur 의 경우는 방금 위에서 설명한 것처럼 focusin, focusout 를 대신하여 사용하면 되므로 사실상 지원되는 것이나 다를 바가 없다. 결국, ready 이벤트만이 live()를 지원하지 않는데, 이는 이벤트 위임이 지원되지 않아도 무관한 이벤트이기에 사실상 모든 이벤트를 다 지원한다고 이야기할 수 있는 것이다.

사실 기존에도 대부분의 이벤트가 live()를 지원하였지만, 1.4 에 들어서 지원이 추가된 이벤트를 나열하자면 다음과 같다.

change, submit, focusin, focusout, mouseenter, mouseleave

다만, 여러분이 인터넷 익스플로러 8 을 사용한다면, change 이벤트를 live()에서 사용하는 경우, 제대로 동작하지 않는 현상이 있을 수 있으므로 주의하기 바란다. 이 부분은 현재 버그 리포팅이 되어 있으며, 마이너 업데이트에서 보완될 것으로 보인다.

8. Queue 관련 기능

새로운 메서드 : .delay()

새로이 추가된 delay() 메서드는 큐 안에서 함수의 실행을 지연시키는 역할을 하는 메서드이다. 일반적으로 애니메이션 효과들 사이에서 시간 지연을 위해 사용하는 편이지만, 사용자 정의 큐가 존재한다면 그러한 큐의 지연을 위해서도 사용이 가능하다. 예를 들어, 다음의 코드를 살펴보자.

```
$('#foo').slideUp(1000).delay(500).fadeIn(1000);
```

이는 foo 라는 id 를 갖는 요소에 대해서 1 초 동안 SlideUp 애니메이션을 수행한 다음, 0.5 초간 쉬었다가, 다시 FadeIn 애니메이션을 1 초간 진행할 것이다. 이로써 훨씬 부드러운 애니메이션을 연출할 수 있을 것이다.

새로운 메서드 : .clearQueue()

clearQueue() 메서드는 현재 큐에 존재하는, 아직 실행되지 않은 모든 함수를 제거한다. 만일, 인자로 해당 큐의 이름을 지정하지 않는다면, 그는 기본적으로 표준 애니메이션 큐(fx)를 대상으로 남아있는 모든 함수를 제거하게 된다. 기능적으로 보았을 때, .stop(true) 메서드와 유사하게 보일 수 있지만, stop() 메서드는 오직 애니메이션에 대해서만 동작하는 반면, clearQueue() 메서드는 .queue() 메서드를 사용하여 구성된 일반적인 큐에 존재하는 모든 함수들도 제거할 수 있다.

9. Ajax 관련 기능

jQuery.param() 메서드의 개선

1.4 부터는 PHP 나 Ruby on Rails 프레임워크를 지원하기 위해서, 중첩 매개변수 직렬화를 지원하고 있다. 즉, jQuery.param() 메서드의 기본 동작이 중첩 매개변수 직렬화를 수행하는 것으로 변경되었으며, 기존의 직렬화 방식을 사용하고 싶다면(권장하지는 않는다), jQuery.param() 메서드의 2 번째 인자로 true 값을 지정해야 한다.

이로 인해, { hobby : ["basketball", "coding"]} 개체가 기존에는 hobby=basketball&hobby=coding 와 같이 직렬화 되었다면, 1.4 부터는 hobby[]=basketball&hobby[]=coding 와 같이 직렬화된다는 것이다.

jQuery.param() 메서드의 기본 동작이 중첩 매개변수 직렬화로 변경되었기에, 기존의 얕은 수준(shallow 방식)의 매개변수 직렬화를 수행하고자 한다면, 앞서 말한 것과 같이 jQuery.param() 메서드의 2 번째 인자로 true 값을 지정해 주어야 한다. 또는, jQuery.ajax() 메서드의 인자 중 traditional 을 true 로 설정하여 ajax 요청 시에 기존 직렬화가 수행되도록 설정할 수도 있다. 만일, 그러한 설정을 매번하는 것이 아니라 한번 설정하여 전역적으로 적용되도록 하고 싶다면, jQuery.ajaxSettings.traditional 를 true 로 설정하여 전역적으로 기존 직렬화를 사용하도록 설정할 수 있다. 다음은 그러한 3 가지 경우를 각각 코드로 보여주고 있다.

```
// 기존 방식으로 stuff 개체에 대한 직렬화를 수행한다.
jQuery.param( stuff, true );
```

```
// 모든 직렬화를 기존 방식으로 동작하도록 전역적으로 설정한다.
jQuery.ajaxSettings.traditional = true;

// 단일 ajax 요청에서 기존 직렬화를 사용한다.
$.ajax({ data: stuff, traditional: true });
```

콘텐트 타입의 자동 탐지(json 과 script 에 대해서)

Ajax 요청에 대한 응답이 json MIME 형식(application/json)이라면, dataType 은 기본적으로 "json"이 된다(만일, dataType 이 지정되지 않았다면). 또한, Ajax 요청에 대한 응답이 JavaScript MIME 형식(text/javascript 또는 application/x-javascript)이라면, dataType 은 기본적으로 "script"가 된다. 기존에는 명시적으로 지정해야만 올바로 동작했었다.

HTML 5 요소 지원

jQuery 1.4 는 최근 이슈가 되고 있는 HTML 5 에 대한 지원도 포함하고 있다. 즉, 다양한 HTML 입력 형식들 을 지원한다는 것이다. 예를 들면, 기존에는 존재하지 않았던 datetime 이나 range 와 같은 입력 컨트롤의 값도 .serialize() 메서드를 통해서 직렬화 된다는 것이다.

Ajax 요청에 대한 컨텍스트의 지원

1.4 부터는 Ajax 요청에 컨텍스트를 지정할 수 있게 되었다. Ajax 요청 시, context 속성을 지정하면, 모든 콜백 메서드에서 해당 컨텍스트를 기반으로 이후의 작업을 수행할 수 있다는 것이다. 기존에는 이를 위해서 클로저 등의 기법을 사용해야 했지만, context 를 사용한다면 훨씬 코드를 간결하게 구성할 수 있다. 다음은 Ajax 요청에 context 를 지정하는 예이다.

```
jQuery.ajax({
    url: "test.html",
    context: document.body,
    success: function(){
        jQuery(this).addClass("done");
    }
});
```

위와 같이 코드를 구성하면, success 콜백 안에서 this 는 document.body 를 참조하게 될 것이다.

jQuery.ajax()의 success 콜백함수 개선

jQuery.ajax() 메서드를 사용하는 경우, success 콜백 함수의 3 번째 인자로 XHR(XMLHttpRequest) 개체를 가질 수 있게 되었다. 기존에는 jQuery.ajax()의 반환 값으로서만 얻을 수 있었던 XHR 개체를 success 콜백에서는 편하게 참조할 수 있게 된 것이다. 다음은 변경된 success 콜백 함수의 시그너처이다.

```
success(data, textStatus, XMLHttpRequest)
```

10. 유틸리티 메서드 관련

새로운 메서드 : jQuery.isEmptyObject()

새로이 추가된 이 메서드는 지정된 개체가 빈 개체인지를 파악하기 위한 유틸리티 메서드이다. 이는 지정된 개체가 어떠한 속성(상속된 속성 포함)도 가지고 있지 않다면 true 를 반환한다. 다음은 사용 예이다.

```
jQuery.isEmptyObject({}) // true
jQuery.isEmptyObject({ foo: "bar" }) // false
```

새로운 메서드 : jQuery.isPlainObject()

지정된 개체가 개체 리터럴(즉, {}를 사용하여 만들어진 개체)이라면 true, 다른 개체 종류이거나 기본 형식의 개체라면 false 를 반환하는 메서드이다. 다음은 사용 예이다.

```
jQuery.isPlainObject({}) // true

function MyClass() {}
jQuery.isPlainObject(new MyClass()) // false
jQuery.isPlainObject(new Date()) // false
```

새로운 메서드 : jQuery.contains()

어떤 DOM 요소 안에 특정 DOM 요소가 포함되어 있는지를 확인하기 위한 메서드이다. 2 개의 인자를 가질 수 있는데, 첫 번째 인자로 지정된 DOM 요소 안에 두 번째 인자로 지정된 DOM 요소가 포함되어 있으면 true, 그렇지 않으면 false 를 반환한다. 즉, 첫 번째 인자가 컨테이너로 사용된다고 생각하면 이해하기가 쉬울 것이다. 각각의 인자로는 반드시 DOM 요소를 지정해야 한다. 다음은 문서 안에 <p> 요소가 존재한다고 가정했을 경우에, 이 메서드를 사용하는 예이다.

```
jQuery.contains($("p")[0], document.body); // false
jQuery.contains(document.body, $("p")[0]); // true
```

보이다시피, 첫 번째 인자로 p 요소(DOM 요소를 접근하기 위해서 $("p")[0] 혹은 $("p").get(0)을 사용한다)를 사용하고 두 번째 인자로 body 를 사용하면, p 요소는 body 요소를 가지고 있지 않으므로 false 가 되고, 반대의 경우는 true 가 되는 것을 볼 수 있다.

새로운 메서드 : jQuery.noop()

1.4 에 추가된 noop() 유틸리티 메서드는 아무 것도 하지 않는 빈 함수가 필요한 경우, 사용할 수 있는 비어있는 함수이다. 실제로 jQuery 1.4.2 의 소스를 살펴보면, noop 함수는 function() { }라고 정의되어 있다. 사실, 이 함수가 정말로 필요한 것인지에 대해서는 의견이 분분하다.

기타 알아두면 좋을 만한 변경사항들

arguments.callee 배제

arguments.callee 는 ECMAScript 5 명세에서 제외될 것으로 예정되어 있기에, jQuery 내부에서 이를 참조하고 있는 코드는 모두 제거되었다.

10. jQuery 플러그인 (from CookBook) part 1

구글을 통한 검색
이전의 방법들은 플러그인의 출처를 알고 있는 경우에 유용한 반면, 구글(Google)을 통해서 웹 전체를 검색하는 것 또한 상당히 유용하다. 다만, 검색된 자료가 상당히 많을 수 있기에, 그만큼 많은 결과를 조사해야 할 필요가 있다. 또한, 구글 질의 검색을 사용한다면, 더 빠르게 찾을 수도 있다.

{searchterm} "jquery*.js"
- 플러그인의 이름이 jquery-{myplugin}.js 또는 jquery.{myplugin}.js인 경우에 최선의 검색 결과가 나온다.

{searchterm} "*jquery.js"
- 플러그인이 명명 규칙을 따르는 경우에 최선의 검색 결과가 나온다.

첫 jQuery 플러그인 작성하기

문제점
jQuery 플러그인을 작성하기로 결정했다고 가정해 보자. 그렇다면, jQuery 로 플러그인을 어떻게 작성해야 할까? 또한, 따라야 하는 권장 규칙은 무엇일까?
해결 방법
jQuery 는 플러그인을 매우 간단하게 그리고 직관적으로 작성할 수 있도록 설계되어 있다. 메서드 또는 함수를 작성하는 것만으로도 기존의 jQuery 개체를 확장할 수가 있으니 말이다. jQuery 코어 라이브러리를 추가한 후에 여러분의 자바스크립트를 선언하면 새로운 사용자 정의 메서드 또는 함수를 사용할 수가 있다.
사용자 정의 jQuery 메서드 작성하기
jQuery 메서드는 체인이 가능하기에 jQuery 셀렉터(selector)의 이점을 활용할 수가 있다. jQuery 메서드를 정의하려면 jQuery.fn 개체에 여러분의 메서드 명을 추가하여 확장하면 된다. jQuery 개체는 여러 결과를 처리할 수 있어야 하기에, 여러분의 코드가 모든 결과에 적용되도록 each() 함수의 호출 내부에 사용자 정의 기능을 두어야만 할 것이다.

```
jQuery.fn.goShop = function() {
  return this.each(function() {
    jQuery('body').append('<div>Purchase: ' + this.innerHTML + '</div>');
  });
};
```

이렇게 만들어진 새로운 플러그인을 사용하는 것은 jQuery 를 호출하는 것만큼이나 쉬운데, 단지 새로운 메서드를 사용하기만 하면 된다.

```
jQuery('p').goShop();
```

사용자 정의 jQuery 함수 작성하기
함수는 jQuery 개체 자체에 추가되며, jQuery 셀렉션의 외부에서 호출되도록 설계되어 있다.

```
jQuery.checkout = function() {
  jQuery('body').append('<h1>Checkout Successful</h1>');
};
```

이 새로운 함수도 다음과 같이 조작 및 호출이 가능하다.

```
jQuery.checkout();
```

논의
jQuery 의 강력한 기능 중 하나는 기본 jQuery 개체에 새로운 메서드와 함수를 추가할 수 있다는 것이다. 그리고, 이미 많은 핵심 메서드들이 이와 동일한 기술을 사용하여 라이브러리에 포함되어 있다. jQuery 에 이미 존재하고 있는 이러한 기능을 활용함으로써, jQuery 사용자들과 플러그인 사용자들은 jQuery 코드를 사용하여 빠르고 간결하게 새로운 기능을 추가하거나, 기존 기능을 확장하거나, 원하는 어떠한 형태로든 만들 수 있게 된다. 이러한 융통성이 바로 jQuery 의 핵심적인 특징이며, jQuery 와 jQuery 플러그인들이 광범위하게 사용되는 이유이다.
새로운 메서드 또는 함수를 통해 jQuery 를 확장하는 결정은 주로 개발자의 필요에 의해 결정된다. 보통, 새로운 메서드의 추가를 통해 jQuery 를 확장시키는 것이 최선의 방법인데, 그 이유는 jQuery 가 새로운 메서드에 대해서도 다른 메서드와의 체인을 지원하기 때문이며, 메서드 안에 있는 코드가 jQuery 셀렉터 엔진을 활용할 수 있도록 허용하기 때문이다.

여러분의 플러그인으로 옵션 전달하기

문제점
여러분이 만든 첫 번째 플러그인은 단지 jQuery 에 메서드를 추가한 것뿐이었다. 그러나, 플러그인에는 적절하게 노출되기만 한다면, 다른 사람들에게 도움이 될만한 약간의 옵션이 있어야 한다. 사용자 정의 메서드에 옵션을 전달하기 위한 가장 좋은 방법은 무엇일까?
해결 방법
결론을 말하자면, 옵션은 options 개체를 통해서 여러분의 사용자 정의 플러그인 메서드로 전달되는 것이 가장 좋다. 메서드의 매개변수로 단일 options 개체를 전달하는 방법이 작업하기에도 쉽고, 보다 정리된 코드를 만들 수 있으며, 유연성도 제공할 수 있다.
플러그인에서 사용할 수 있는 옵션을 제공하는 경우에는, 적절한 기본값들을 제공하는 것이 바람직하다. 또한, 기본값들을 재정의할 수 있는 메서드를 제공하는 것 또한 중요하다. 이렇게 하려면, 기본 옵션 개체를 선언한 다음, extend() 메서드를 사용하여 기본값을 사용자가 지정한 값으로 재정의하면 된다. 그리고 나면, 여러분의 코드에서 이 옵션들을 사용할 수 있다.

```
jQuery.fn.pulse = function(options) {
  // 기본 옵션들과 전달된 옵션들을 병합한다.
  var opts = jQuery.extend({}, jQuery.fn.pulse.defaults, options);

  return this.each(function() {
    // Pulse 기능이 시작된다.
    for(var i = 0;i < opts.pulses;i++) {
      jQuery(this).fadeTo(opts.speed,opts.fadeLow).fadeTo(opts.speed,opts.fadeHigh);
    }

    // 원래의 상태로 재설정한다.
    jQuery(this).fadeTo(opts.speed,1);
  });
};

// Pulse 플러그인의 기본 옵션들이다.
jQuery.fn.pulse.defaults = {
  speed: "slow",
  pulses: 2,
  fadeLow: 0.2,
  fadeHigh: 1
};
```

기본 옵션이 지정되었기에, 플러그인을 사용하는 개발자들이 함수를 호출하면서 많게든 적게든 옵션을 사용할 수 있게 되었다. 플러그인의 주요 메서드를 사용하자마자 기본값들을 설정하는 것은 중요하며, 그렇지 않으면 오류가 발생할 수도 있다.

```
// 하나의 옵션만을 재정의한다.
jQuery('p').pulse({pulses: 6});

// 모든 옵션들을 재정의한다.
jQuery('p').pulse({speed: "fast", pulses: 10, fadeLow: 0.3, fadeHigh: 0.8});
```

마지막으로, 플러그인 함수에 연결된 자식 개체에 옵션을 지정하는 방식으로, 기본 옵션을 프로젝트에서 한번 더 재 정의할 수 있다. 그리고 나면, 개발자들은 자신의 기본 옵션 집합을 지정할 수 있게 되며, 원하는 동작을 만들어내는 데에 필요한 코드 양을 줄일 수 있게 된다.

```
// 위에는 플러그인 코드가 존재하고 있다.
```

```
// Pulse 플러그인의 기본 옵션들을 재설정한다
jQuery.fn.pulse.defaults = {
    speed: "fast",
    pulses: 4,
    fadeLow: 0.2,
    fadeHigh: 1
};

// 이번 호출에서는 새로운 기본 옵션들을 사용할 것이다
jQuery('p').pulse();
```

논의
플러그인에서 옵션을 제공하는 것은 플러그인에 상당한 유연성을 추가할 수 있는 효과적인 방법이다. 풍부한 옵션 집합을 제공하는 플러그인은 더 광범위한 지지자들의 요구에 부합할 것이고, 다양한 작업들을 수행할 수 있을 것이며, 옵션을 제공하지 않는 플러그인보다 더 많은 인기를 얻게 될 것이다.
여러분의 플러그인에 기본 옵션 집합을 포함시키는 것은 플러그인을 사용하는 개발자들에게 플러그인이 구현된 방식에 따라 선택권과 유연성을 제공하는 또 다른 수단이 된다. 또한, 유리한 점으로는 플러그인이 항상 정의되어 있는 특정 옵션들에 의존적이기에 옵션이 전달되었는지 여부를 검사하기 위한 코드가 필요 없어지며 그에 따라 코드의 양이 감소된다는 것이다. 결국, 플러그인을 호출할 때마다 하나의 옵션이나 여러 개의 옵션, 심지어는 모든 옵션들을 재정의할 수 있는 능력을 플러그인 사용자들에게 제공하는 셈이다. 마지막으로, jQuery 개체에 기본 옵션을 추가하였기에, 옵션은 전역적으로 재정의될 수도 있다. 이는 새롭고 창의적인 방식으로 활용할 수 있는 또 다른 방안을 플러그인 사용자에게 제공하는 것이 된다.

11. jQuery 플러그인 (from CookBook) part 2

플러그인에서 $ 단축표현 사용하기

문제점
다른 자바스크립트 라이브러리들은 $라는 단축표현을 사용한다. jQuery 도 jQuery 라고 명명된 기본 개체와 더불어 $를 유일한 단축 표현으로써 사용하고 있다. 여러분의 플러그인이 다른 플러그인이나 라이브러리들과 함께 사용되는 경우, 호환성을 유지할 수 있다고 어떻게 보장할 수 있을까?
해결 방법
jQuery 는 jQuery 개체를 위한 사용자 정의 별칭으로써 $ 함수를 사용하고 있다. 하지만, jQuery 가 호환성 모드로 설정되면, jQuery 는 $을 정의하고 있는 다른 라이브러리에게로 $ 별칭에 대한 제어권을 양보한다. 플러그인도 동일한 기법을 사용하도록 작성될 수 있다.
다음과 같이 익명 함수로 여러분의 플러그인을 둘러싼 뒤 곧바로 그 함수를 실행하는 방식으로, $ 단축 표현을 플러그인 내부에서 유지되게 할 수 있다. 물론, 플러그인 외부 코드에서도 $를 사용할 수는 있지만, 플러그인 코드 내부에 있는 $만이 jQuery 개체를 참조할 것이다.

```
;(function($) {
  $.fn.pulse = function(options) {
    // 기본 옵션들과 전달된 옵션들을 병합한다
    var opts = $.extend({}, $.fn.pulse.defaults, options);

    return this.each(function() {
      // Pulse 기능이 시작된다
      for(var i = 0;i<opts.pulses;i++) {
        $(this).fadeTo(opts.speed,opts.fadeLow).fadeTo(opts.speed,opts.fadeHigh);
      }

      // 원래의 상태로 재설정한다
      $(this).fadeTo(opts.speed,1);
    });
  };

  // Pulse 플러그인의 기본 옵션들이다
```

```
  $.fn.pulse.defaults = {
    speed: "slow",
    pulses: 2,
    fadeLow: 0.2,
    fadeHigh: 1
  };
})(jQuery);
```

논의

익명 함수로 여러분의 코드를 둘러싸는(wrap) 것은 여러분의 플러그인 코드를 보다 광범위한 영역에서 훌륭하게 동작하게 할 뿐만 아니라 여러 기능들을 추가하기에 매우 직관적이며 간결한 방법이라 할 수 있다.
함수 정의의 시작 부분에 세미콜론(;)을 추가하는 것은 다른 개발자들이 개발한 라이브러리의 종료 부분에 세미콜론이 누락된 상황을 대비하기 위한 것이다. 자바스크립트 언어는 기본적으로 줄 바꿈(newline)을 통해 문장을 구분하곤 하지만, 많은 사용자들이 프로젝트에 있는 자바스크립트들을 단일 파일로 압축시키는 최소화 도구를 이용하는 편이다. 이러한 도구를 사용하게 되면 줄 끝이 제거되기에, 여러분의 코드가 바로 이후에 올 경우 오류가 발생할 가능성이 있다. 따라서, 처음 부분에 세미콜론을 추가하여 그러한 문제를 빠르고 간단하게 막을 수 있는 것이다.
이전의 코드는 괄호(()를 사용하여 곧바로 익명 함수에 대한 정의를 시작하고 있다. 그리고, 익명 함수 안에서는 jQuery 개체 대신에 사용할 변수를 인자로 전달하도록 함수를 정의하고 있는데, 그 변수로는 $를 사용하고 있다. 이와 같이 추가적인 익명 함수를 정의하는 것은 자바스크립트 언어가 영역을 처리하는 방식 때문에 필요한 것이기도 하다. 자바(Java)와 C++과 같은 전통적인 언어에서의 영역(scope)은 블록 문으로 제한되지만, 자바스크립트에서의 영역은 함수로서 형성된다. 그러므로, 이전의 코드에서 함수를 사용한 이유는 사실 플러그인 내부에 정의할 수 있는 영역의 경계를 설정하기 위한 것이다.
이어지는 코드는 새로운 버전의 플러그인 코드이며, 기존과 비교했을 경우 유일하게 변경된 것은 jQuery 개체 대신 단축 표현인 $를 사용한다는 것이다. 익명 함수를 사용해서 플러그인을 래핑하고, $ 변수의 영역을 제한하고 있기에, 이제 어떠한 다른 코드와의 충돌 없이 자유롭게 $를 사용할 수 있다.
마지막 줄은 각각 영역 함수를 닫는 괄호(})와 익명 함수를 닫는 괄호())로 마무리 되고 있으며, 익명 함수의 정의를 마친 후에 즉시 그를 실제로 호출하고 있다. 이는 함수 내에서 $로 명명되는 jQuery 개체를 전달하겠다고 익명 함수에게 통보하는 것과 같다. 끝으로, 자바스크립트 최소화와 압축으로 인한 문제를 미연에 방지하기 위해서 세미콜론(;)을 사용하여 문장을 끝내고 있다.
$ 단축 표현은 자바스크립트 코드를 작성하는 데 있어 대단히 유용하다. $ 단축 표현을 사용한다면 코드의 크기를 줄일 수 있을 뿐만 아니라 훌륭한 코드 설계를 이끌 수도 있다. 또한, $ 단축 표현은 대단히 대중적일 뿐만 아니라 잘 알려져 있다. 그래서, 많은 라이브러리들이 $ 단축 표현을 이용하고 있으며, 그들의 컨텍스트에 $ 단축 표현을 포함시키고 있다. 그렇기 때문에, $ 단축 표현에 대한 자신들만의 버전을 제공하는 각각의 라이브러리들을 사용하는 경우 서로 충돌이 날 가능성이 상당히 높다. 따라서, 익명 함수 내부에 여러분의 플러그인 코드를 래핑시키는 것은 $ 단축 표현을 사용하는 영역의 수준을 관리할 수 있게 하여, 다른 자바스크립트 라이브러리를 사용하는 경우에 발생할 수 있는 잠재적인 충돌을 감소시키는 효과가 있다.
익명 함수로 여러분의 플러그인을 래핑할 경우 얻을 수 있는 추가적인 효과는 클로저(closure)가 생성된다는 것이다. 자바스크립트에서 클로저를 사용하면, 여러분이 정의할 필요가 있는 모든 메서드 및 변수를 적절하게 네임스페이스화 하는데 도움을 줄 수 있으며, 더 나아가 변수명 또는 함수명이 다른 코드와 충돌할 가능성을 줄여준다.

플러그인에 전용(private) 함수 포함하기

문제점
플러그인 코드가 비대해져서, 체계적으로 코드를 구성해야 할 필요가 있다고 가정해 보자. 그 경우, 플러그인 외부에서는 접근할 수 없는 코드인 전용(private) 메서드를 어떻게 하면 구현할 수 있을까?
해결 방법
레시피 12.4 에서부터 시작된 플러그인 설계 패턴을 이용하는 경우, 전용 함수는 일반적으로 플러그인을 감싸고 있는 익명 함수에 정의될 수 있다. 전용 함수가 익명 함수 안에 들어있기 때문에, 외부 코드에서는 전용 메서드를 확인할 수가 없다. 그러므로, 외부 코드에서는 jQuery 개체에 추가된 함수 또는 메서드만을 볼 수 있다.

```
;(function($) {
  $.fn.pulse = function(options) {
    // 기본 옵션들과 전달된 옵션들을 병합한다
    var opts = $.extend({}, $.fn.pulse.defaults, options);

    return this.each(function() {
      doPulse($(this),opts);
    });
  };
```

```
  function doPulse($obj,opts) {
    for(var i = 0;i<opts.pulses;i++) {
      $obj.fadeTo(opts.speed,opts.fadeLow).fadeTo(opts.speed,opts.fadeHigh);
    }

    // 원래의 상태로 재설정한다
    $obj.fadeTo(opts.speed,1);
  }

  // Pulse 플러그인의 기본 옵션들이다
  $.fn.pulse.defaults = {
    speed: "slow",
    pulses: 2,
    fadeLow: 0.2,
    fadeHigh: 1
  };
})(jQuery);
```

논의
현재 익명 함수로 둘러싸인 플러그인을 가지고 있기에, 플러그인 내부에 private 함수를 정의하는 것은 평소처럼 새로운 함수를 추가하는 것만큼이나 간단하다.
공개(public) 메서드와 전용(private) 메서드를 사용하여 플러그인을 그룹화하고 체계화하는 것은 사용자들과 플러그인 작성자들에게 많은 이점을 제공할 수 있다. 플러그인을 완성한 후 커뮤니티로부터 플러그인에 대한 피드백을 받음으로써, 플러그인 버전 별로 일관된 API 를 제공하도록 공개 메서드와 전용 메서드의 사용을 조절할 수 있다. API 의 일관성은 플러그인의 성공에 있어 중요한 요인 중 하나이다.
또한, 전용 메서드와 공개 메서드로 코드를 분해하는 것은 여러분의 플러그인의 규모가 커짐에 따라 코드 구성에 있어 중요한 이점을 가지게 된다. 잘 짜여진 코드는 읽기 쉽고, 유지 보수하기도 쉬우며, 테스트를 하기에도 용이하다. 그리고, 깔끔한 코드는 에러가 발생할 확률도 줄여줄 것이다.

Metadata 플러그인 지원하기

문제점
몇몇 플러그인들은 플러그인 메서드에 사용자 정의 옵션을 전달하기 위해서 Metadata 플러그인을 사용한다. Metadata 플러그인과 함께 구성되도록 하려면 어떻게 통합시켜야 할까?
해결 방법
Metadata 플러그인을 사용하려면, 우선 그 플러그인을 이용할 수 있는지를 검사한 다음에 메타데이터 매개변수들을 사용하여 플러그인 옵션들을 확장하면 된다. 즉, 플러그인을 호출하는 경우에는 기본 옵션을 제공하고, 실행 시에 마크업에 작성된 메타데이터를 사용하여 각 개체의 기본 옵션을 재정의할 수 있다는 것이다.

```
<!-- Metadata 플러그인을 포함시킨다 -->
<script type="text/javascript" src="metadata/jquery.metadata.js"></script>

<!-- 메타데이터를 포함하고 있는 마크업의 예제는 다음과 같다 -->
<p class="{pulses: 8, speed: 'slow'}">Starship Enterprise</p>
<p>Battlestar Galactica</p>
<p class="{speed: 100}">Serenity</p>


;(function($) {
  $.fn.pulse = function(options) {
    // 기본 옵션들과 전달된 옵션들을 병합한다
    var opts = $.extend({}, $.fn.pulse.defaults, options);
```

```
    return this.each(function() {
      // 특정 노드에 있는 메타데이터 요소를 병합한다
      var o = $.metadata ? $.extend({}, opts, $.metadata.get(this)) : opts;
      doPulse($(this),o);
    });
  };

  function doPulse($obj,opts) {
    for(var i = 0;i<opts.pulses;i++) {
      $obj.fadeTo(opts.speed,opts.fadeLow).fadeTo(opts.speed,opts.fadeHigh);
    }

    // 원래의 상태로 재설정한다
    $obj.fadeTo(opts.speed,1);
  }

  // Pulse 플러그인의 기본 옵션들이다.
  $.fn.pulse.defaults = {
    speed: "slow",
    pulses: 2,
    fadeLow: 0.2,
    fadeHigh: 1
  };
})(jQuery);
```

논의

Metadata 플러그인을 포함시키는 것은 jQuery 플러그인들이 어떻게 함께 사용될 수 있는지를 보여주는 좋은 예제라 할 수 있다. jQuery 플러그인 생태계는 광범위하기에, 얼마든지 다른 플러그인들을 사용할 수 있다.

Metadata 플러그인을 사용하기 위해서는, 먼저 여러분의 스크립트에 Metadata 플러그인을 포함시켜야만 한다. Metadata 플러그인은 구글 코드(Google Code)에 jQuery 와 마찬가지로 호스팅되고 있다. Metadata 플러그인은 여러분의 HTML 에 추가적으로 삽입한 데이터를 이용하여 동작한다. 이전의 코드에서는 항목의 class 요소 안에다가 요소에 특화된 옵션들을 넣고 있으며, 그를 메타데이터로 사용하고 있다.

옵션은 표준 JSON 을 사용하여 HTML 에 삽입되어 있으며, 모든 옵션들이 지정되거나 아예 지정되지 않을 수도 있다. 이는 사실 플러그인을 사용하는 개발자들이 결정할 일이다. Metadata 플러그인이 제공하는 몇몇 메서드와 옵션은 Metadata 플러그인 설명서 페이지(http://docs.jquery.com/Plugins/Metadata) 에서 확인할 수 있다. 이전의 예제에서는, 먼저 Metadata 플러그인이 포함되어 있는지를 검사하고 있다. 그 이유는 이러한 추가적인 기능을 선택적으로 사용할 수 있도록 하기 위해서, 그리고 하위 호환성을 지원하기 위해서이다. Metadata 플러그인은 단일 요소에 대해 동작하기 때문에, 옵션을 처리하는 방법을 나누어 살펴봐야 한다. 첫 번째 단계는 플러그인이 호출된 시점에 제공된 옵션들을 사용하는 것이다. 이러한 옵션들은 기본 옵션들을 기반으로 하여 확장되며, 이는 이미 기존 플러그인에 반영이 되어 있다. 그리고, 두 번째 단계는 각각의 요소에 정의되어 있는 메타데이터를 사용하여 이러한 기본 옵션들을 확장하는 것이다. 그렇기에, 결국 우리에게 필요한 작업은 메타데이터 옵션을 사용하여 기본 옵션들을 확장하는 것이다.

Metadata 플러그인도 플러그인 자신의 옵션으로 지정할 수 있는 또 다른 옵션을 제공하고 있다. 사용자들에게 옵션을 제공한다는 것은 여러분의 플러그인이 jQuery 생태계의 훌륭한 일원이 되어가고 있다는 것을 보여주는 훌륭한 행동이다. 또한, Metadata 플러그인은 HTML 요소에 사용자 정의 옵션들을 삽입하는 방식을 사용하기에, 상대적으로 적은 코드를 작성하게 하는 좋은 방식이라 할 수 있다.