**0. jQuery**

* 가볍고 빠르며 간결한 오픈소스 자바스크립트 라이브러리
* Prototype, ExtJs등과 유사한 스크립트 라이브러리이다.
* jQuery를 이용하면 Rich 웹 UI를 개발하는 데 도움이 되는 다양한 기능들(HTML 문서 트래버스[순회탐색], 이벤트 처리, 애니메이션, Ajax 처리)을 지원하여 빠르고 견고하게 Rich 웹 애플리케이션을 개발 할 수 있다.
* 특징
  + 막강한 CSS 셀렉터
  + 크로스 브라우저 지원
  + 메서드 체인
  + Ajax 지원
  + 풍부한 플러그인 지원
* $(document).ready();
  + 문서의 DOM 요소들을 조작 가능한 시점이 되면 자동으로 호출이 되는 이벤트 메서드
  + window.onload 이벤트와 유사한 것
  + 메서드의 인자로는 델리게이트 함수명을 기입하거나, 익명 함수를 작성

**1. Selectors**

* jQuery 셀렉터
  + jQuery의 가장 강력한 부분은 HTML DOM을 마음대로 순회 탐색할 수 있다는 것이다. CSS 셀렉터를 사용하여 원하는 개체를 탐색할 수 있다.

CSS 셀렉터 : 아래와 같은 css 파일에서 사용했던 표현식

div p {  
    font-color:red;  
}  
  
#loginID {  
    font-weight:bold;  
    background:yellow;  
}  
  
.Columns {  
    padding:10px;  
    background:white;  
}

div p라는 셀렉터는 현재 문서 상에서 div 요소의 자식으로 존재하는 모든 p 요소들에 적용

#loginID는 loginID라는 ID값을 가진 요소 적용

.Columns는 class 어트리뷰트 값으로 Columns를 갖는 모든 요소에 적용

* + jQuery에서 이러한 셀렉터를 그대로 사용 가능
  + $(selector) 혹은 jQuery(selector)과 같은 표현식을 가진다.
  + $(“div p”), $(“#loginID”), $(“.Columns”)

각각의 표현식은 각 DOM 요소의 확장 개체인 jQuery 개체 집합을 반환한다. DOM 요소를 직접 반환해주는 것이 아니라 그의 래퍼인 jQuery 개체로 반환해주기 때문에 직접 DOM 요소를 제어할 때보다 훨씬 편하고 쉽게 개체를 제어할 수 있다. 예를 들어 $(“div p”).hide() 와 같은 명령을 사용하면 현재 문서 상에서 div 요소의 자식으로 존재하는 모든 p태그 집합들은 눈에서 보이지 않게 된다. jQuery 셀렉터에 의해 반환되는 개체가 일반 DOM 개체라면 이러한 명령을 사용할 수 없겠지만 jQuery 개체 집합으로 반환되기에 그러한 명령을 사용할 수 있게 되는 것입니다.

* + 이러한 기본적인 CSS 셀렉터 외에 고급 CSS 셀렉터도 지원한다. 예를 들면, 계층 셀렉터, 일반 필터 셀렉터, 어트리뷰트 필터 셀렉터 등을 지원한다.

- P > a : p 요소 바로 아래 자식인 a 요소와 일치된다.

- div + p : div 요소의 바로 다음에 나오는 형제 p 요소와 일치된다.

- div ~ p : div 요소의 다음에 나오는 모든 형제 p 요소와 일치된다.

* + $(“p a”)와 $(“p > a”)의 차이점은 전자가 p요소 하위에 존재하는 모든 a 요소를 선택한다면, 후자는 p 요소 바로 아래의 자식으로 놓여있는 a요소만을 선택한다는 차이가 있다.
  + 더불어, a[title]이나 a[href^="mailto:"]와 같이 어트리뷰트를 기반으로 하는 필터링도 가능하다. 특정 어트리뷰트가 존재하거나, 그 어트리뷰트 값이 특정 값으로 시작 혹은 끝나거나, 특정 값을 포함하거나 하는 부분까지 비교해서 선택할 수 있다. 다음은 이러한 어트리뷰트 필터의 예입니다.

|  |  |
| --- | --- |
| 셀렉터 | 설명 |
| \* | 모든 요소와 일치 |
| E1 | E1(태그명)인 모든 요소와 일치 |
| E1.class | E1(태그명) 요소의 클래스가 class와 동일한 요소와 일치 |
| E1.#id | E1(태그명) 요소의 id 어트리뷰트 값이 id와 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 E2와 일치 |
| E1 ~ E2 | E1 요소의 다음에 나오는 모든 형제 E2와 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 |

a[title] : title 어트리뷰트를 갖는 하이퍼 링크와 일치된다.  
a[href^="mailto:"] : href 값이 mailto로 시작하는 하이퍼 링크와 일치된다.  
a[href$=".pdf"] : pdf 파일에 링크가 걸린 모든 하이퍼링크와 일치된다.  
a[href\*="taeyo.net"] : taeyo.net이라는 값이 포함되어 있는 모든 하이퍼 링크와 일치된다.  
input[type="text"] : text 형식의 입력 컨트롤과 일치된다.

데모 : Default.html

* jQuery 셀렉터 : 필터
  + 기본적인 셀렉터에 더하여 jQuery는 다양한 셀렉터 필터들도 제공합니다.
  + 위치를 기반으로 필터링하거나 하위 자식들을 필터링할 수 있게 한다거나 컨텐츠를 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.
  + 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(0부터 시작) |
| :odd | 홀수 요소들과 일치합니다(0부터 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 |

tr:odd : 현재 문서에 존재하는 모든 tr 중 홀수번재 열들만을 선택  
tr:gt(4) : 문서에 존재하는 모든 tr중 그 인덱스가 4 이상인 모든 열들을 선택  
tr:last : 모든 tr중 가장 마지막 tr을 선택하는 셀렉터  
:header:eq(1) : 문서에 존재하는 헤더들 중에서 인덱스가 1번째인 헤더를 선택

데모 : filter01.html

* + 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. |

b:contains('j')  : “j”라는 단어를 포함하는 모든 b 요소들을 선택  
div:has('ul')  : 모든 div 요소들 중에 ul 요소를 가지고 있는 div만을 선택  
table.nameTable td:empty  : nameTable이라는 css 클래스가 지정된 table들 중에서 자식 요소인 td 내부에 텍스트 값이 없는(또는 자식요소가 없는) 모든 td를 선택합니다.  
.nameTable td:contains('F')  : nameTable이라는 css 클래스가 지정된 요소들 중에서 그 내부의 td가 텍스트로 ‘F’란 단어를 포함하고 있는 모든 td를 선택  
table.nameTable tr:eq(0) : nameTable이라는 css 클래스가 지정된 table 내의 tr중에서 첫번째 tr을 선택

데모 : filter02.html

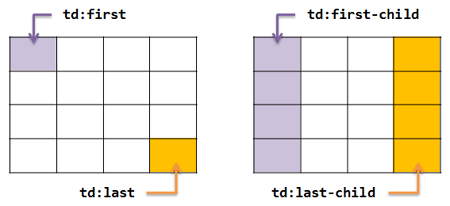
* + 자식 필터

|  |  |
| --- | --- |
| :nth-child(index/even/odd) | 자식 중 index로 지정된 위치의 요소들과 일치되거나, even, odd에 해당하는 자식들과 일치됩니다. 단, 여기서의 index는 1부터 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 |

: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-child얼핏 보기에는 같은 요소를 선택할 것처럼 보이는 이 둘은 확실한 차이를 보입니다. 즉,

td:first는 현재 테이블 안에서 첫 번째로 나오는 td 하나 만을 선택하는 반면, td:first-child는 현재 테이블 안에서 나오는 첫 번째 수준의 td들을 모두 선택한다는 것이죠.



* + 폼필터

|  |  |
| --- | --- |
| :input | 모든 input, textarea, select, button요소들과 일치됩니다 |
| :text | text 타입의 모든 input 요소들과 일치됩니다. |
| :password | password타입의 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. |

**2. Attributes**(<http://api.jquery.com/category/attributes/>)

|  |  |
| --- | --- |
| attr( *name* ) | 메치된 첫 번째 요소의 특정 어트리뷰트에 접근하여 값을 가져온다. 만일, 지정된 어트리뷰트 명이 존재하지 않는다면 undefined가 반환된다. 어트리뷰트에는 title, alt, src, href, width, style와 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 |

샘플 : attr.html

사용자 정의 어트리뷰트라는 것은 다음과 같이 임의의 어트리뷰트를 여러분이 설정하는 것을 말합니다.

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

상기 태그에서 status와 order이라는 어트리뷰트가 바로 사용자 정의 어트리뷰트 되겠습니다.

샘플 : attr2.html

**3. Traversing(탐색메서드)**(<http://api.jquery.com/category/traversing/>)

<필터링>

|  |  |
| --- | --- |
| 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 true가 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. |

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");

가장 간단하게 예를 들면, 다음 2개의 표현은 결과적으로는 동일하다고 볼 수 있습니다.

$("div:odd")   
$("div").filter(":odd")

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/false를 알려주는 메서드인데요. 예를 들어, 다음의 코드를 한번 살펴보시죠.

var $myDiv = $("div").eq(5);  
if ($myDiv.is("div"))   
{  
  $myDiv.css("border", "4px solid yellow");  
}

위의 소스는 $myDiv 변수에 6번째 div를 참조한 뒤, is(expr) 메서드를 사용해서 그 개체가 "div" 개체인지를 비교하여, 만일 "div"가 맞다면, 그 div에 대해서는 노란색의 두꺼운 테두리를 갖도록 지정하고 있지요.

샘플 : traverse.html

<찾기와 관계된 API>

|  |  |
| --- | --- |
| find(expr) | 지정된 표현식과 일치하는 요소를 검색한다. |
| add(expr) | 현재 요소에 expr와 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 |

add(expr) 메서드는 셀렉터에 의해 1차적으로 검색된 결과 집합에 추가적인 검색 집합을 합치는(add) 역할을 합니다.

 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.

샘플 : 4.html

**4. Manipulation**(조작관련메서드:<http://api.jquery.com/category/manipulation/>)

요소에 값을 지정한다거나, 특정 요소의 값을 읽어온다거나 하는 작업을 포함해서, 동적으로 요소 자체를 생성, 삭제, 복사, 제거하는 기능들을 의미합니다.

<요소의 내용을 다루는 메소드>

|  |  |
| --- | --- |
|  | |
| html() | 일치된 요소의 html 내용을 가져옵니다. 이는 요소의 innerHTML 값과 동일합니다. 만일, 일치된 요소가 여러 개라면 그 중 첫 번째 요소의 HTML을 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. |

text() 메서드는 개체의 컨텐트가 html 요소들을 포함하고 있다고 해도, 다 무시하고 텍스트만을 반환한다. 즉, <span><b>taeyo</b></span> 와 같은 span 개체가 있을 때, 이 span 개체에 대해서 text()를 수행하면, 반환값은 내부 html을 제외한 "taeyo"가 되고요. html() 메서드를 사용하면 <b>taeyo</b>를 반환한다는 것이죠.'

<특정요소, 개체 집합을 다른 요소의 "내부"에다가 앞, 뒤로 덧붙이는 기능을 제공하는 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. |

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

이는 #linkText 라는 아이디를 갖는 요소를 얻어서, 그를 link라는 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 <font>(클릭)</font>라는 태그 문자열을 동적으로 생성한 뒤, 그를 모든 B 태그의 뒤에 덧붙이는 역할을 합니다.

<요소 외부에 추가하는 메서드>

|  |  |
| --- | --- |
|  | |
| after(*content*) | 일치된 요소 뒤에 *content*를 삽입합니다. 요소 내부가 아닌 외부에 삽입된다는 것을 제외하면 append와 동일합니다. |
| before(*content*) | 일치된 요소 앞에 *content*를 삽입합니다. 요소 내부가 아닌 외부에 삽입된다는 것을 제외하면 prepend와 동일합니다. |
| insertAfter(*selector*) | 일치된 요소를 *selector*에 의해 일치된 모든 요소들 뒤쪽에 삽입합니다. 요소 내부가 아닌 외부에 삽입된다는 것을 제외하면 appendTo와 동일합니다. |
| insertBefore(*selector*) | insertBefore(*selector*)와 유사하나, 요소 앞쪽에 삽입합니다. 요소 내부가 아닌 외부에 삽입된다는 것을 제외하면 prependTo와 동일합니다. |

$("a").after("<font>(클릭)</font>");

모든 하이퍼링크 바로 뒤에 "<font>(클릭)</font>" 이라는 태그 문자열을 덧붙이는 구문입니다. 결과로 다음과 같은 태그가 구성된다.

<a href="http://xxx">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://xxx">사이트<font>(클릭)</font></a>

<삭제용 메서드와 복사용 메서드>

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

복사 관련 메서드(clone)는 인자로서 true를 지정하게 되면, 단순히 요소만을 복사하는 것이 아니라, 그 요소에 달려있는 이벤트 처리기(예, click, mouseover)등도 복사가 된다. 인자가 없는 기본 clone() 메서드로 복사하게 되면, 요소 자체만 복사될 뿐, 해당 요소에 달려있는 이벤트 처리기들은 복사가 되지 않는다.

샘플 : manip.html

**5. CSS 관련 메서드**(<http://api.jquery.com/category/css/>)

|  |  |
| --- | --- |
| 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 true를 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. |

샘플페이지 : css.html

**6. Events**(<http://api.jquery.com/category/events/> )

jQuery가 제공하는 이벤트 관련 메서드들

|  |  |
| --- | --- |
| bind( type, data, fn ) | 매치된 요소에 이벤트 처리기를 바인딩한다. type에는 이벤트 명칭을, data에는 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. |

$("#MyBtn").bind("click", function() {  
   alert("누가 내 버튼을 클릭한 것이냐?");  
});

사용 가능한 이벤트명 : blur, focus, load, resize, scroll, unload, beforeunload, click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout, mouseenter, mouseleave, change, select, submit, keydown, keypress, keyup, error

$("#MyBtn").unbind("click");

jQuery는 일회성 이벤트 처리를 지원하기 위한 one 이라는 메서드도 제공해 줍니다. 이는 이벤트 처리기를 딱 한번만 실행시키고 그 이상으로는 실행시키지 않는 일회성 매핑입니다.

$("#MyBtn").one("click", function() {  
    alert("누가 내 버튼을 클릭한 것이냐?");");  
});

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.

$("#MyBtn").trigger("click");

이 코드는 사용자가 버튼을 클릭한 것처럼 클릭 이벤트를 일으키게 합니다. 코드를 사용해서 강제로 이벤트를 발생시키는 방법이라는 것이죠.

bind( type, data, fn )

두 번째 인자인 data는 선택적인 인자입니다. 즉, bind 메서드는 총 인자가 2개가 들어오면 내부적으로 첫 번째 인자를 type으로 인식하고, 두 번째 인자는 fn 으로 인식하게 됩니다. 그리고, 총 3개의 인자가 지정되는 경우에만 첫 번째 인자를 type으로, 두 번째 인자를 data로, 그리고 세 번째 인자를 fn으로 인식하는 것이죠. 해서, 앞의 예제에서는 인자를 2개만 전달해도 잘 동작했던 것입니다.

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.

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

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.

샘플 : EventEx1.html

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

샘플 : EventEx2.html

**7. Effects**(<http://api.jquery.com/category/effects/> )

샘플문서

**8. AJAX**

타입1.

$.ajax({

type: "GET",

url: "test.js",

dataType: "script"

});

동적으로 js파일을 로드해서 실행시키는 기능을합니다.

타입2.

$.ajax({

type: "POST",

url: "some.php",

data: "name=John&location=Boston",

success: function(msg){

alert( "Data Saved: " + msg );

}

});

some.php 에 POST 타입으로 name 과 location 을 parameter 로 넘기는 형태. success 프라퍼티는 서버에서 http response 를 받은후에 수행될 콜백함수를 지정합니다.

타입3.

$.ajax({

url: "test.html",

cache: false,

success: function(html){

$("#results").append(html);

}

});

test.html 을 가장 최근에 수정된 페이지로 받겠다는 형식

타입4.

var html = $.ajax({

url: "some.php",

async: false

}).responseText;

async(비동기) 라는 프라퍼티가 false이므로 "동기"로 요청한다는 형식

some.php로 request를 날린후에 response가 오기전까지는 브라우저를 블락킹 시킨다.

응답 데이터는 다음과 같이 받는다.

var html = $.ajax().responseText;

**9. Utilities**(<http://api.jquery.com/category/utilities/>)

예제 참고

**10. jQuery UI**