jQuery

jQuery

- 전체 웹사이트의 70% 이상이 사용하고 있는 **라이브러리**
- 라이브러리이지 자바스크립트와 별개의 언어가 아님.

단점

- 무겁다.
- 제이쿼리 없이 순수 자바스크립트만 사용한 코드보다는 속도가 10에서 100배정도 느림
- 요즘은 React, Angular 사용.

장점

- 코드가 보기 좋다.
- 웹디자이너분들처럼 디자인을 위해 제이쿼리를 많이 사용함.

- document.getElementById('zero')
- $('#zero')

- 하나의 코드로 여러 브라우저에서 다 호환되는 것이 진짜 큰 **장점**
- **플러그인**을 지원해서 원하는 기능을 추가

jQuery Download

1. 제이쿼리 홈페이지에 가서 파일로 다운받아서 홈페이지 안에 넣는 방법
2. **cdn**을 사용하는 방법
3. **bower**이나 **npm**같은 패키지 매니저를 사용하는 방법

- 제일 간단하고 속도도 괜찮은 cdn을 사용

cdn

<script src="//code.jquery.com/jquery-latest.min.js"></script>

- jquery1.11.1이 설치

jQuery selector

- 제이쿼리를 선택하는 가장 큰 이유가 제이쿼리 선택자입니다.
- css 선택자를 그대로 사용할 수 있어 코드 선택이 매우 간단

- *아이디(#)*
- *클래스(.)*
- *태그*
- *속성([])*
- *자식 선택자*
- *후손 선택자*
- *형제자매 선택자*
- *가짜 선택자*
- **조합**

아이디(#)

- document.getElementById('zero');

→ $('#zero');

#을 앞에 붙여 zero라는 id를 선택

ES5 대안
document.querySelector('#zero')

클래스(.)

- document.getElementsByClassName('zero');
- $('.zero');
- document.querySelectorAll('.zero')

Tag

- document.getElementsByTagName('p');
- $('p');

속성([])

- 자바스크립트에선 id도 아니고 class도 아니고, 태그이름도 아니어서 태그의 속성으로 검색을 해야함,
- 제공되는 함수가 없어서 개발자가 직접만들어서 처리.

- Jquery는 제공
- $('[value]'); // value라는 속성을 가진 태그
- $('[value="zero"]');

- ES5
- document.querySelectorAll('[value]')
  document.querySelectorAll('[value="zero"]')

자식 선택자 : 자식 태그는 해당 태그 바로 아래의 태그

- . #zero 태그의 자식인 p 태그를 선택

```
Array.prototype.filter.call(document.getElementById('zero').children,function(item) {
      return item.tagName === 'P';
  });
```

위 코드를 jquery로 바꾸면

```
$('#zero > p');
```

→ > 기호가 자식 태그임을 알려줌.

ES5

```
document.querySelectorAll('#zero > p');
```

후손 선택자: 후손이란 해당 태그 아래에 포함된 모든 태그

- #zero 태그 아래의 모든 div를 선택
- document.getElementById('zero').getElementsByTagName('div');
- $('#zero div')
- document.querySelectorAll('#zero div');

형제자매 선택자

- + 는 다음 하나만 선택
- ~는 형제자매 중 일치하는 것을 모두 선택

- $('#zero + div'); // #zero 바로 다음 div를 선택합니다.
- $('#zero ~ p'); // #zero의 형제자매 p를 모두 선택합니다.

- document.querySelectorAll('#zero + div');
- document.querySelectorAll('#zero ~ p');

조합

- 좋은 점은 위의 선택자들을 조합
- 태그 이름이 p면서 id는 zero고 checkbox인 태그를 선택
- - $('p#zero:checkbox');

- 여러 개의 선택자를 동시에 선택

- $('#zero, .nero

- #zero와 .nero 그리고 p 태그를 모두 선택, p')

jQuery DOM traversing

html
--head
----title
--body
----div#container
------header
------div.main
--------nav
--------div#zero
----------p
------------b
------------em
----------span
----------input
--------div#nero
--------div#hero
------footer

DOM traversing

- DOM을 자유자재로 이동하는 방법

- $('div#zero') → 시작점 선택

- **parent**
  - $('div#zero').parent(); // [div.main]
    document.getElementById('zero').parentElement;

DOM traversing

- **parents**
  - 부모부터 시작해서 조상 태그를 모두 선택

  - $('div#zero').parents(); // [div.main, div#container, body, html]

DOM traversing

- **parentsUntil**
  - 부모부터 시작해서 선택한 조상 태그 이전까지를 모두 선택

  - $('div#zero').parentsUntil('body'); // [div.main, div#container]

DOM traversing

- **closest**
  - 조상 태그를 선택할 수 있습니다. #container을 선택하고 싶다면

  - $('div#zero').closest('#container'); // [div#container]

DOM traversing

- **children**
  - 자식 태그를 모두 선택
  - $('div#zero').children(); // [p, span, input]
  - document.getElementById('zero').children;

DOM traversing

- **first**
  - 선택한 태그 중 첫 번째 태그
  - $('div#zero').children().first(); // [p]
  - document.getElementById('zero').firstChild;

DOM traversing

- **last**
  - 선택한 태그 중 마지막 태그를 선택
  - $('div#zero').children().last(); // [input]
  - document.getElementById('zero').lastChild;

DOM traversing

- **eq**
  - 선택한 태그 중 n번째 태그를 선택합니다.
  - 순서가 0, 1, 2, 3, ...이기 때문에 첫 번째 것은 0, 두 번째 것은 1 순번

  - $('div#zero').children().eq(1); // [span]

  - document.getElementById('zero').children[1];

DOM traversing

- **find**
  - 후손 태그를 선택
  - $('div#zero').find('em'); // [em]
  - document.getElementById('zero').getElementsByTagName('em')

DOM traversing

- **next**
  - 형제자매 태그 중 다음 태그를 선택
  - $('div#zero').next(); // [div#nero]
  - document.getElementById('zero').nextElementSibling;

DOM traversing

- **nextAll**
  - 형제자매 태그 중 다음 모든 태그를 선택

  - $('div#zero').nextAll(); // [div#nero, div#hero]

DOM traversing

- **nextUntil**
  - 다음 태그부터 선택한 태그 이전까지를 모두 선택

  - $('div#zero').nextUntil('div#hero'); // [div#nero]

DOM traversing

- **prev**
  - 형제자매 태그 중 이전 태그를 선택

  - $('div#zero').prev(); // [nav]
  - document.getElementById('zero').prevElementSibling;

DOM traversing

- **prevAll**
  - 형제자매 태그 중 이전 모든 태그를 선택
  - $('div#nero').prevAll(); // [nav, div#zero]
- **prevUntil**
  - 선택한 태그 이후부터 이전 태그까지를 모두 선택
  - $('div#nero').prevUntil('nav'); // [div#zero]

DOM traversing

- **Siblings**
  - prevAll과 nextAll을 합쳐놓은 메소드입니다. 모든 형제자매 태그를 선택

  - $('div#zero').siblings(); // [nav, div#nero, div#hero]

- **Filter**
  - 선택한 태그들 중에서 원하는 태그를 걸러냄
  - $('div').filter('#zero'); // [div#zero]

메소드 체이닝 기법

- 위의 메소드들은 연달아 사용할 수 있습니다

- $('div#zero').prev().parent().next().children().eq(0)

속성, 내용, 스타일, 데이터 조회 및 변경

- Content

- get 메소드는 제이쿼리 객체를 자바스크립트 객체로 다시 전환하는 역할을 합니다. get 메소드 안의 인자는 여러 개의 태그(클래스같은 경우 여러 개의 태그를 동시에 선택 가능) 중에 몇 번째 태그를 자바스크립트 객체로 전환할 건지 선택할 수 있게 해줍니다.

- $('#zero').get(0); //

- attr 속성을 조회하거나 변경할 수 있는 메소드입니다. 인자를 하나만 주면 해당 속성의 값을 조회하고, 두 개를 주면 해당 속성의 값을 변경

- $('#zero').attr('name'); // 'zero'
- $('#zero').attr('name', 'hero'); // name 속성을 hero로 변경
- document.getElementById('zero').name = 'hero';

- value 속성만을 다루는 메소드입니다. 인자가 없으면 value를 조회하고, 인자가 있으면 그 값으로 value를 바꿉니다.

- $('#zero').val(); // 'Hello‘
- $('#zero').val('hero'); // value를 hero로 변경
- document.getElementById('zero').value = 'hero'

- prop 메소드는 attr과 유사하지만 속성중에 true/false 값을 가지는 속성들만을 위한 메소드입니다.
- 대표적인 예로 hidden, readonly, checked 등의 속성이 있습니다.

- $('#zero').prop('hidden'); // false
- $('#zero').prop('hidden', true); // hidden 속성의 값을 true로 변경
- document.getElementById('zero').hidden = true;

- **addClass**
  - 클래스를 추가하는 메소드입니다.

- $('#zero').removeClass('genius');
- document.getElementById('zero').classList.remove('genius');

- **removeClass**
  - 클래스를 제거하는 메소드입니다.

- $('#zero').removeClass('genius');
- document.getElementById('zero').classList.remove('genius');

- **toggleClass**
  - 해당 클래스가 있으면 지우고, 없으면 만듭니다.

- $('#zero').toggleClass('genius'); // genius 추가
- $('#zero').toggleClass('genius'); // genius 삭제
- document.getElementById('zero').classList.toggle('genius')

- **html**
  - innerHTML과 같습니다.

- $('#zero').html(); // Content
- $('#zero').html('Changed'); // 내용을 Changed로 변경
- document.getElementById('zero').innerHTML = 'Changed';

- **text**
- textContent를 바꿉니다. innerHTML과는 달리 태그를 넣을 수는 없습니다.

- $('#zero').text(); // Content
- $('#zero').text('TextChanged'); // 내용을 TextChanged로 변경

- **css**
  - 스타일 속성을 조회하거나 변경합니다. 인자를 하나만 주면 해당 스타일 속성 값을 조회하고, 두 개를 주면 해당 값으로 변경합니다. 두 개의 인자를 따로 주는 대신에 하나의 객체를 인자로 넘겨 여러 스타일을 동시에 바꿀 수 있습니다.

- $('#zero').css('width'); // 465px
- $('#zero').css('width', 300); // 300px로 변경
- $('#zero').css({ width: 300, height: 300 }); // 여러 스타일 동시 변경

- document.getElementById('zero').style.width = '300px';

- **height**
  - 태그의 높이를 알려줍니다.
  - ('#zero').height();
  - document.getElementById('zero').style.height;

- **width**
  - 태그의 너비를 알려줍니다.
  - $('#zero').width();
  - document.getElementById('zero').style.width;

- **position**
  - 태그의 상대적인 위치를 알려줍니다.
  - $('#zero').position(); // { left: 값, top: 값 }
  - var el = document.getElementById('zero');
  - { left: el.offsetLeft, top: el.offsetTop }

- **data**
  - 마지막으로 태그에 데이터를 조회하거나 저장하는 방법입니다. html5에서는 태그에 데이터를 저장할 수 있습니다. data-age, data-name 처럼 **data- 접두어**를 붙이면 됩니다. 이렇게 만들어진 데이터는 data 메소드로 가져올 수 있습니다.

- $('#zero').data('age'); // 23
- $('#zero').data('age', 24); // 24로 data-age를 24로 변경
- document.getElementById('zero')['data-age'] = 24;

jQuery utils

- 제이쿼리에는 코드를 관리하는 데 도움을 주는 유틸들이 내장 되어 있습니다. 굳이 선택자를 통해 태그를 선택하지 않고도 바로 쓸 수 있습니다.

- **$.each**
  - 배열이나 객체를 반복하는 메소드입니다. ES5의 forEach이나 for ~ in 문과 비슷합니다. 함수의 첫 번째 인자는 **키**, 두 번째 인자는 **값**입니다.

```
$.each(['zero', 'hero', 'nero'], function(key, val) {
    console.log(key, val);
 }); // 0, zero, 1, hero, 2, nero

$.each({ zero: 'me', hero: 'friend', nero: 'enemy' }, function(key, val) {
    console.log(key, val);
}); // zero, me, hero, friend, nero, enemy

['zero', 'hero', 'nero'].forEach(function(val, key) {
    console.log(val, key);
}); // zero, 0, hero, 1, nero, 2
```

- **$.extend**
- 첫 번째 객체에 다른 객체들의 속성을 복사하는 메소드입니다. ES2015의 Object.assign과 비슷합니다.

- $.extend({ zero: 'genious' }, { zero: 'babo', hero: 'friend' }); // { zero: 'babo', hero: 'friend' });

- Object.assign({ }, { zero: 'babo' }); // { zero: 'babo' }

- **$.trim**
  - 문자열의 좌우 공백을 제거하는 유틸입니다. 문자열.trim() 메소드와의 차이점은 거의 없어 보입니다.

- $.trim(' zero '); // zero

- ' zero '.trim(); // zero

- **$.inArray**
  - 배열 안에 해당 요소가 몇 번째에 있는지 검사하는 메소드입니다. 없으면 -1을 반환합니다. 배열.indexOf 메소드와 비슷합니다.

  - var arr = ['zero', 'hero', 'nero'];
  - $.inArray('hero', arr); // 1
  - $.inArray('jero', arr); // -1
  - arr.indexOf('zero'); // 0

- **$.proxy**
- this를 바꾸는 메소드입니다. ES5에서의 bind 메소드와 비슷합니다.

- var func = function() {
  console.log(this);
  };
- var obj = { zero: 'babo' };
- func(); // window
- var newFunc = $.proxy(func, obj);
- newFunc(); // obj
- var bindFunc = func.bind(obj);
- bindFunc; // obj

- **$.type**
  - 기존에 있던 **typeof** 연산자를 조금 보완한 것입니다. typeof 연산자는 배열과 null, Date 등을 모두 object라고 표시합니다. 또한 new를 통해서 만든 문자열, 숫자, 불린도 모두 object라고 표시하고요. 정규표현식도 브라우저마다 다르게 처리합니다. 제이쿼리의 $.type은 위의 것들을 구분해줍니다.

- $.type(new Date); // date
- $.type([1, 2, 3]); // array
- $.type(null); // null
- $.type(/s/); // regexp

태그 생성, 추가, 복사, 이동, 제거

- **생성**
  - 순수 자바스크립트와 비교하면 정말 간단합니다.
  - $안에 만들고 싶은 태그를 적어넣으면 됩니다.

  - var $div = $(' Hello ');
  - var div = document.createElement('div');
  - var text = document.createTextNode('Hello');
  - div.appendChild(text);

- **append, appendTo**
  - 위에서 만든 태그를 추가해봅시다. 맨 마지막 자식 요소로 추가됩니다. *부모.append(자식)* 또는 *자식.appendTo(부모)*처럼 하면 됩니다.

  - $('#zero').append($div);
  - $div.appendTo($('#zero'));
  - document.getElementById('zero').appendChild(div);

- **prepend, prependTo**
  - 맨 마지막 요소로 추가하는 게 싫다면 맨 첫 요소로 추가할 수도 있습니다. *부모.prepend(자식)* 또는 *자식.prependTo(부모)*처럼 하면 됩니다.

  - $('#zero').prepend($div);
  - $div.prependTo($('#zero'));

  - document.getElementById('zero').insertBefore(div,document.getElementById('zero').firstChild);

- **after, insertAfter**
  - 특정 태그 다음에 추가할 수도 있습니다. *부모.after(자식)* 또는 *자식.insertAfter(부모)* 하면 됩니다.

  - $('#zero').after($div);
  - div.insertAfter($('#zero'));

  - document.getElementById('zero').parentElement.insertBefore(div, document.getElementById('zero').nextElementSibling);

- **before, insertBefore**
  - 특정 태그 이전에 추가하는 방법입니다. *부모.before(자식)* 또는 *자식.insertBefore(부모)* 하면 됩니다.

  - $('#zero').before($div);
  - $div.insertBefore($('#zero'));

  - document.getElementById('zero').parentElement.insertBefore(div, document.getElementById('zero'));

- **clone**
  - 특정 태그를 복사합니다. 태그 안의 내용까지 전부 다 복사됩니다.

  - var $zero = $('#zero').clone(); // $zero에 복사된 태그가 담김
  - var zero = document.getElementById('zero').cloneNode(true);

- **remove, detach**
  - 특정 태그를 제거합니다. remove는 완전히 제거해버리지만, detach는 잠시 제거하는 겁니다. 변수에 저장해두었다가 나중에 appendTo로 다시 붙이면 됩니다.
  - var $zero = $('#zero').detach();
  - $('#zero').remove();

  - document.getElementById('zero').parentNode.removeChild(document.getElementById('zero'));

- **empty**
  - 태그 안의 내용을 비웁니다.
  - $('#zero').empty();
  - document.getElementById('zero').innerHTML = '';

제이쿼리 이벤트

- 제이쿼리의 장점 중 하나.
- 이벤트 관리가 편리함.
- 선택한 태그에 이벤트를 붙이기만 하면 됨.

- **$(function() { 내용 });**
- 이 코드의 역할은 document가 ready되면 콜백 함수의 내용을 실행하라는 겁니다. ready 되었다는 것은 DOM 로딩이 완료되었다는 겁니다. (이미지나 아이프레임은 제외합니다)

- $(function() { alert('hello') });

```
function ready(fn) {
	if (document.readyState != 'loading'){
		fn();
	} else {
		document.addEventListener('DOMContentLoaded', fn);
	}
}
ready(function() { alert('hello'); });
```

- **On, Off, One**
- on 메소드가 대표적입니다. bind나 live, delegate 메소드를 쓰시는 분이 있는데 안 쓰는 것을 **권장**합니다. 모든 기능이 on으로 통합되었습니다. on 메소드는 다음과 같이 사용합니다. 태그에 on 메소드를 붙여주면 됩니다.

```
$('#zero').on('이벤트명', function() {
        // 내용...
});
```

- 여러 개의 이벤트를 동시에 달 수 있습니다.
- 첫 번째 것은 여러 개의 이벤트에 하나의 콜백 함수만 다는 경우
- 두 번째는 각각의 이벤트에 다른 콜백 함수를 달아주는 겁니다.
- 세 번째는 두 번째와 같은데 그냥 이벤트를 **체이닝** 기법으로 연달아 달아주는 겁니다.

```
$('#zero').on('이벤트1 이벤트2', function() {
        // 내용...
});

$('#zero').on({
        이벤트1: function() {},
        이벤트2: function() {},
        ...
});

$('#zero').on('이벤트1', function() { ... }).on('이벤트2', function() { ... })...
```

- 아직 생성되지 않은 태그에 이벤트를 미리 달아놓을 수도 있습니다. 만들어지지도 않았는 데 왜 이벤트를 미리 달아놓을까요? 예를 들면 **AJAX** 요청으로 태그를 새로 가져오거나 만들 때가 있습니다. 이런 태그들에 이벤트를 직접 달기 어려울 때, 이벤트를 미리 달아놓는 기법을 사용합니다. 이를 Delegation이라고 부르며, 이전의 delegate 메소드가 이 역할을 했다가, on으로 통합되었습니다.

  - 두 번째 인자로 선택자를 받는 부분이 추가되었습니다. 선택자 부분에 새로 추가될 태그의 선택자를 추가하는 겁니다.
  - $(document).on('이벤트명', '선택자', function() { // 내용; });

  - 달았던 이벤트를 끄려면 off 메소드를 사용합니다.
  - $('#zero').off('이벤트명')

- 만약 같은 이벤트가 여러 개 달려있어서 선택적으로 끄고 싶다면, 이벤트 이름 뒤에 네임스페이스를 붙여줄 수 있습니다. 아래의 예는 click 이벤트가 두 개 달려있습니다. 두 클릭 이벤트를 구분하기 위해 점 뒤에 네임스페이스를 달아줬습니다. first와 second가 그것입니다. 나중에 네임스페이스를 사용하여 off로 특정 이벤트만 끌 수 있습니다.

  - $('#zero').on('click.first', function() { ... }).on('click.second', function() { ... });
  - $('#zero').off('click.first'); // click.second 이벤트는 유지

  - 이벤트를 한 번만 실행되게 하려면 on 메소드 대신 one 메소드를 사용합니다.

  - $('#zero').one('이벤트명', function () {
    // 내용...
    });

이벤트 메소드

- on 외에도 직접 이벤트명을 메소드로 하는 메소드들이 있습니다. click, mouseenter, mouseleave, dbclick, input, ready 등이 거의 모든 이벤트에 해당하는 메소드들이 있습니다만 저는 그냥 on 메소드를 쓰는 것을 추천합니다. 나중에 수정하기 편합니다.

- $('#zero').click(function() {});
- $('#zero').mouseenter(function() {});
- $('#zero').hover(function() {});

  -> mouseenter 와 mouseleave합쳐놓은 메소드. 즉, 마우스를 어떤 태그위에 올렸다 뺐다 할 때 수행할 동작을 정의할때 사용

이벤트 객체

- 콜백 함수의 첫 인자로는 이벤트 객체가 전달
  - $('#zero').click(function(event) {
    event.prevenDefault();
    event.stopPropagation();
    });

- **preventDefault** 메소드는 태그의 기본 동작(예를 들면, a 태그는 클릭 시 링크이동, form 태그은 폼 내용 전송)을 하지 않게 막아주는 역할을 합니다.
- **stopPropagation** 메소드는 태그를 클릭 시 부모에게 이벤트가 전달되지 않도록 합니다.

이벤트 실행

- trigger 메소드로 이벤트를 실행할 수 있습니다.
  - $('#zero').on('click', function() { 내용 });
  - $('#zero').trigger('click');

- trigger 메소드의 장점은 커스텀 이벤트를 만들고 호출할 수 있다
  - $('#zero').on('custom', function() { 내용 });
  - $('#zero').trigger('custom');

기타

- 다른 특별한 점으로는 콜백 함수 안에서 $(this)로 이벤트가 발생한 객체를 선택할 수 있습니다.

```
$('a').click(function() {
    $(this); // 여러 개의 a태그 중 클릭된 a 태그를 선택
});
```

- 이벤트 과정 도중에 데이터를 전달할 수 있습니다.
- 전달한 데이터는 event 객체의 data 속성에 들어있습니다.

```
$('#zero').on('click', { name: 'Zero' }, function(event) {
    alert(event.data.name); // Zero
});
```

애니메이션

- 제이쿼리는 또 애니메이션이 편리하기로 유명함.
- 많은 웹 디자이너분들이 제이쿼리를 사용.

- **hide, show, toggle**

  - 태그를 숨겼다 보였다 하는 메소드입니다. show하면 보여지고, hide하면 사라집니다. 첫 번째 인자로 숫자를 넣어 보여지고 숨겨지는 시간을 설정할 수 있습니다. 단위는 ms입니다. 또는 slow(600ms), normal(400ms), fast(200ms) 문자열을 넣어 속도를 조절할 수도 있습니다. 그리고 함수를 인자로 넣으면 애니메이션이 끝났을 때 동작합니다.
  - toggle은 보이는 상태에서는 사라지게 하고, 사라진 상태에서는 보이게 합니다. 즉 두 가지 상태를 전환합니다. 스위치 같은 메소드입니다.

  - $('#zero').show();
  - $('#zero').hide();
  - $('#zero').show(1000); // 1000ms -> 1초
  - $('#zero').hide('slow');
  - $('#zero').toggle(1000, function() {
    alert('짜잔!'); //1초동안 나타나거나 사라진 다음 '짜잔!'하고 alert
    });

- **slideUp, slideDown, slideToggle**

  - slideUp은 슬라이드쇼가 올라가는 효과를 보이며 태그가 사라집니다. slideDown은 슬라이드쇼가 내려가듯 태그가 나타납니다.
  - slideToggle은 두 상태를 전환합니다. 기본 속도는 400ms입니다. 마찬가지로 끝나고 일어날 동작을 정의할 수 있습니다.

  - $('#zero').slideUp();
  - $('#zero').slideDown('fast');
  - $('#zero').slideToggle(500, function() {
    alert('스르륵');
    });

- **fadeIn, fadeOut, fadeToggle, fadeTo**

  - fadeIn은 서서히 태그가 나타납니다.
  - fadeOut은 서서히 태그가 사라집니다.
  - fadeToggle은 두 상태를 전환합니다.
  - fadeTo 메소드도 있습니다. 이 메소드는 특정 불투명도(opacity)로 태그의 불투명도를 조정합니다.

  - $('#zero').fadeOut();
  - $('#zero').fadeIn('normal');
  - $('#zero').fadeToggle(800, function() {
    alert('사사삭');
    });
  - $('#zero').fadeTo(500, 0.5); // opacity를 0.5로 서서히 바꿉니다.

- **delay, stop**

  - 애니메이션 실행을 조금 늦추거나(delay),
  - 중간에 멈출 수 있습니다(stop);

  - $('#zero').delay(500).hide(); // 500ms 기다렸다가 hide
    $('#zero').show().stop(); // show를 멈춤

- **animate**
  - 애니메이션의 끝판왕입니다. 애니메이션을 원하는 대로 정의할 수 있습니다. 사실 위의 fadeIn, slideUp 등도 다 내부적으로 animate를 사용합니다. 인자로 객체를 제공합니다. left 속성은 '+=100'으로 되어있고, height는 100으로 고정되어 있습니다.
  - left는 현재 left 값에서 100을 더한 값으로 움직이고, height는 100으로 움직입니다. 두 번째 인자로는 애니메이션이 실행될 시간을 ms로 전달하고, 세 번째 인자로는 애니메이션이 완료된 후의 행동을 함수로 전달할 수 있습니다.

  - ```
    $('#zero').animate({
            left: '+=100',
            height: 100

     }, 500, function() {
          console.log('애니메이션 완료!');
    });
    ```
    ```

- **queue, dequeue**
  - 큐는 실행될 애니메이션 개수에 대한 정보를 갖고 있고, 애니메이션이 끝났을 때의 콜백을 전달할 수 있습니다. 아래의 예는 위와 같습니다. dequeue는 queue가 끝났다는 것을 제이쿼리에게 알려줄 수 있습니다.

  - ```
    $('#zero').animate({
        left: '+=100',
        height: 100
    }, 500)
      .queue(function() {
            console.log('애니메이션 완료!');
            $('#zero').dequeue();
      });
    ```

    큐는 애니메이션의 개수에 대한 정보를 갖고 있습니다.
    queue 안에 fx(또는 queue의 이름)를 넣고 length 속성을 조회하면 몇 개의 애니메이션이 대기중인지 나옵니다.

    ```
    $('#zero').queue('fx').length
    ```
    ```

AJAX

asynchronous Javascript and XML

- AJAX가 jQuery 고유의 기능은 아닙니다.
- 하지만 jQuery는 간단한 문법으로 AJAX를 사용할 수 있게 도와줌

- 기존의 웹에서는 한 번 페이지를 로딩하면 다른 페이지를 로딩하기 위해서 링크를 타고 넘어가야 했습니다. 그렇게 되면 흔히 말하는 페이지
- 깜빡임이 발생합니다.

- 이름처럼 비동기적으로 서버에 요청을 하여 페이지 전환 없이도 새로운 데이터를 가져올 수 있습니다. 로딩 표시가 뜰 때 뒤에서는 데이터를 가져오고 있는 겁니다.

AJAX 작동 방식

- 브라우저 내장 객체인 XMLHttpRequest 객체를 통해 웹 서버에서 데이터를 요청합니다.
- 자바스크립트 및 HTML DOM을 통해 데이터를 표시하거나 사용합니다.

1.웹 페이지에서 이벤트 발생(페이지가 로드되고 버튼이 클릭됨)
2. XMLHttpRequest 객체는 JavaScript에 의해 생성됩니다.
3. XMLHttpRequest 개체는 웹 서버에 요청을 보냅니다.
4. 서버가 요청을 처리합니다.
5. 서버는 웹 페이지에 응답을 다시 보냅니다.
6. JavaScript로 응답을 읽습니다.
7. 적절한 조치(예: 페이지 업데이트)는 JavaScript에 의해 수행됩니다.

Fetch API

- 최신 브라우저는 XMLHttpRequest 객체 대신 Fetch API 를 사용할 수 있습니다.
- Fetch API 인터페이스를 사용하면 웹브라우저가 웹 서버에 HTTP 요청을 할 수 있습니다.
- XMLHttpRequest 객체를 사용하는 경우보다 Fetch는 더 간단한 방법으로 동일한 작업을 수행할 수 있습니다.

XMLHttpRequest 객체

- AJAX의 핵심은 XMLHttpRequest 객체입니다. 우선적으로 XMLHttpRequest 객체를 생성한 후 콜백 함수를 정의하고 XMLHttpRequest를 열어 서버에 요청을 보냅니다.

- 모든 최신 브라우저는 XMLHttpRequest 객체를 지원합니다. 웹서버와 데이터를 교환하는데 사용하고, 전체 페이지를 다시 로드하지 않고, 웹페이지의 일부를 업데이트할 수 있습니다.

XMLHttpRequest 객체 생성 구문

variable = new XMLHttpRequest();

콜백함수 정의

xhttp.onload = function() { *//응답이 준비되면 수행할 작업* }

- 콜백함수는 다른 함수의 매개변수로 전달되는 함수입니다.
- 이 경우 콜백 함수에는 응답이 준비되었을 때 실행할 코드가 포함되어야 합니다.

요청 보내기

xhttp.open("GET", "읽어올 파일경로");
xhttp.send();

- 서버에 요청을 보내려면 XMLHttpReques객체의 open() 및 send() 메서드를 사용할 수 있습니다.
- 보안 상의 이유로 최신 브라우저는 도메인 간 액세스를 허용하지 않습니다.
- 즉, 로드하려는 웹페이지와 XML, JSON 등의 파일이 모두 동일한 서버에 있어야 합니다.