

Modern Web Design

DOM Javascript & jQuery 2013

Speaker : ◦ ㅌㅁ @yamoo9

에이젝스? 아작스???


AJAX ?!

Asynchronous Javascript And XML

AJAX 비동기 통신 기술

February 18, 2005

Ajax: A New Approach to Web Applications

BY JESSE JAMES GARRETT

0 Comments

If anything about current interaction design can be called “glamorous,” it’s creating Web applications. After all, when was the last time you heard someone rave about the interaction design of a product that wasn’t on the Web? (Okay, besides the iPod.) All the cool, innovative new projects are online.

Despite this, Web interaction designers can’t help but feel a little envious of our colleagues who create desktop software. Desktop applications have a richness and responsiveness that has seemed out of reach on the Web. The same simplicity that enabled the Web’s rapid proliferation also creates a gap between the experiences we can provide and the experiences users can get from a desktop application.

That gap is closing. Take a look at [Google Suggest](#). Watch the way the suggested terms update as you type, almost instantly. Now look at [Google Maps](#). Zoom in. Use your cursor to grab the map and scroll around a bit. Again, everything happens almost instantly, with no waiting for pages to reload.

Google Suggest and Google Maps are two examples of a new approach to web applications that we at Adaptive Path have been calling Ajax. The name is shorthand for Asynchronous JavaScript + XML, and it represents a fundamental shift in what’s possible on the Web.

Defining Ajax

AJAX

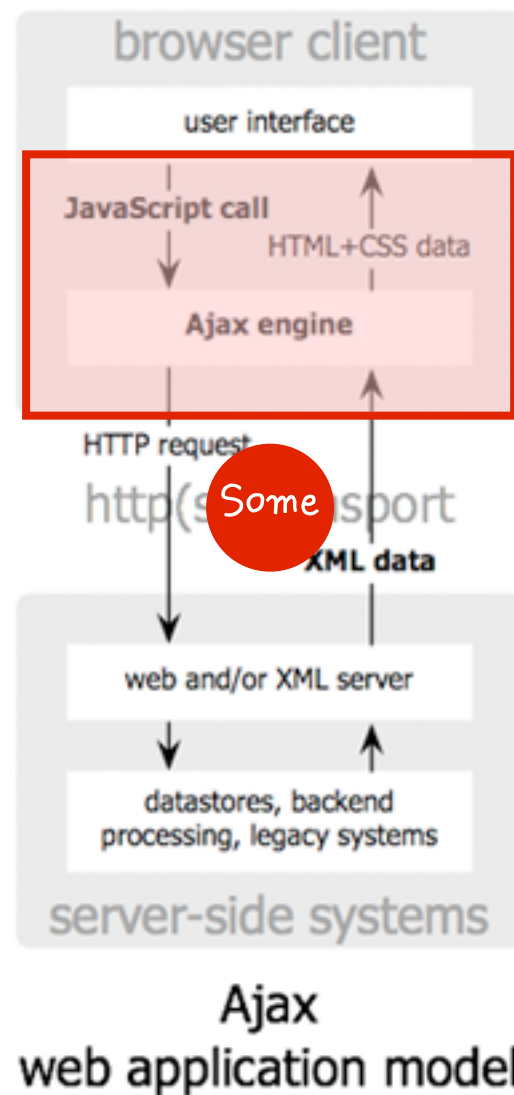
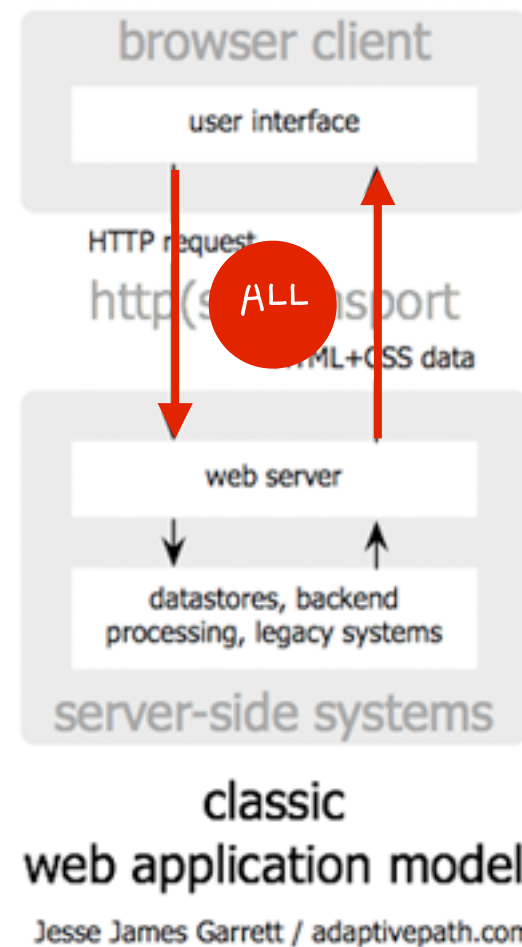


Adaptive Path, CCO
Jesse James Garrett

AJAX는 새로운 기술이 아니죠.

특정한 기술이 아닌, 기술들 간의
연계를 묶어 사용하는 용어입니다.

하지만 제가 작성한 기사 글로 인해
AJAX는 대중화 됩니다.



Adaptive Path, CCO
Jesse James Garrett

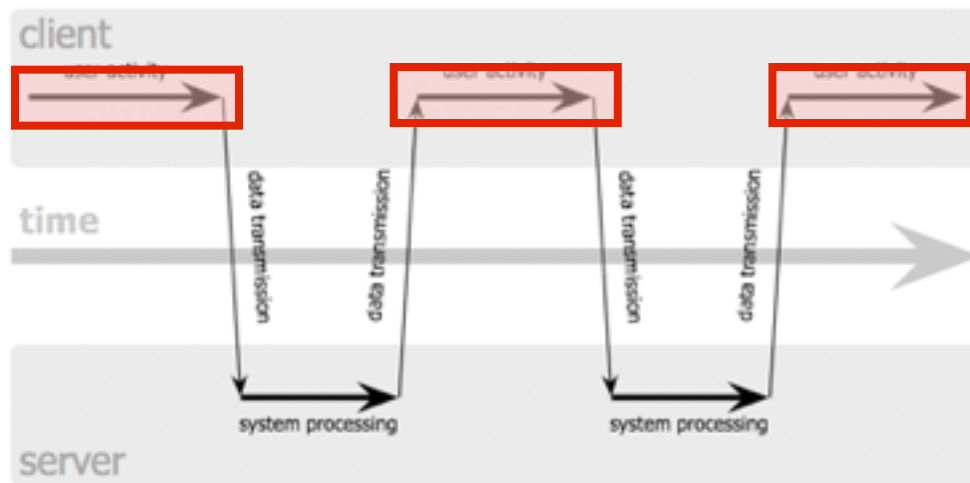
AJAX가 기존 기술과 다른 점은..
요청/응답 과정을 통해 불필요한 부분
까지 처리하지 않는다는 점입니다.

쉽게 말해 필요한 부분만 별도로 요청
하고, 응답 받아 처리할 수 있는 거죠.

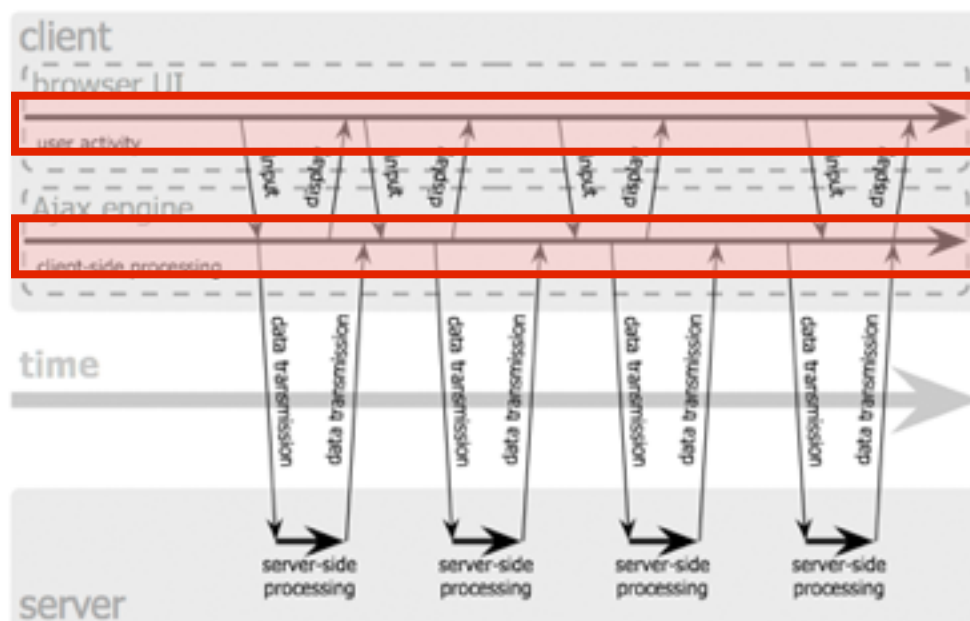
모든 데이터를 업데이트 할 필요가 없
으니, 불필요한 대역폭 감소가 가능하
고, 이는 비용 절감을 가져옵니다.

AJAX

classic web application model (synchronous)



Ajax web application model (asynchronous)



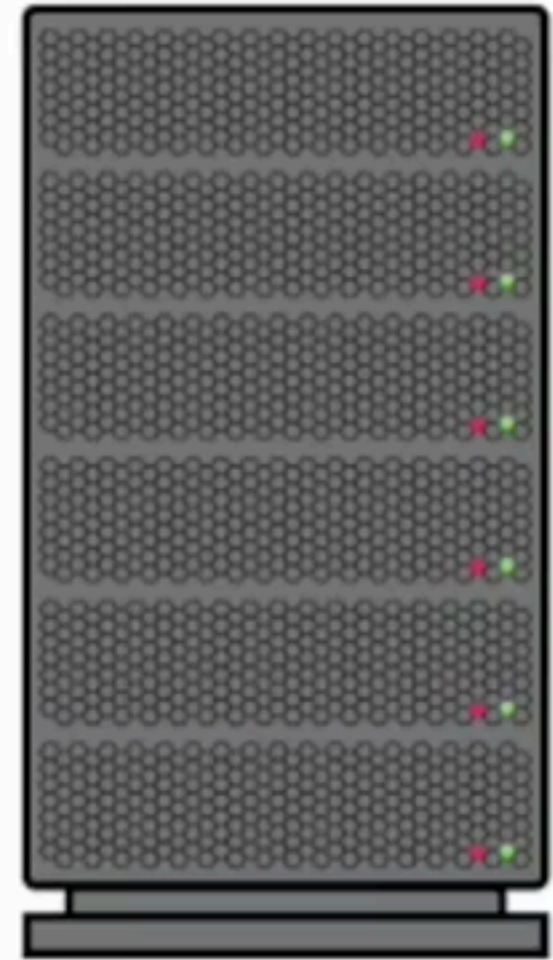
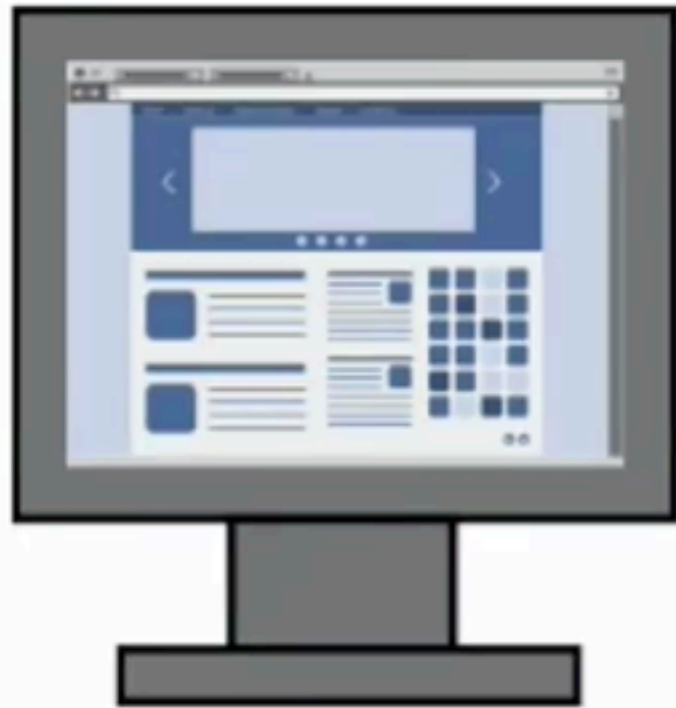
Adaptive Path, CCO
Jesse James Garrett

AJAX의 또다른 장점은 ...

사용자가 대기하는 시간을 줄일 수 있다는
점이죠. 페이지를 새로고침 하지 않고
도 필요한 데이터만 받아와서 내용을
업데이트 시킬 수 있으니까요.

이것이 가능한 이유는 기존의 동기 방식
이 아닌, 비 동기 방식으로 데이터를
요청/처리하기 때문입니다.

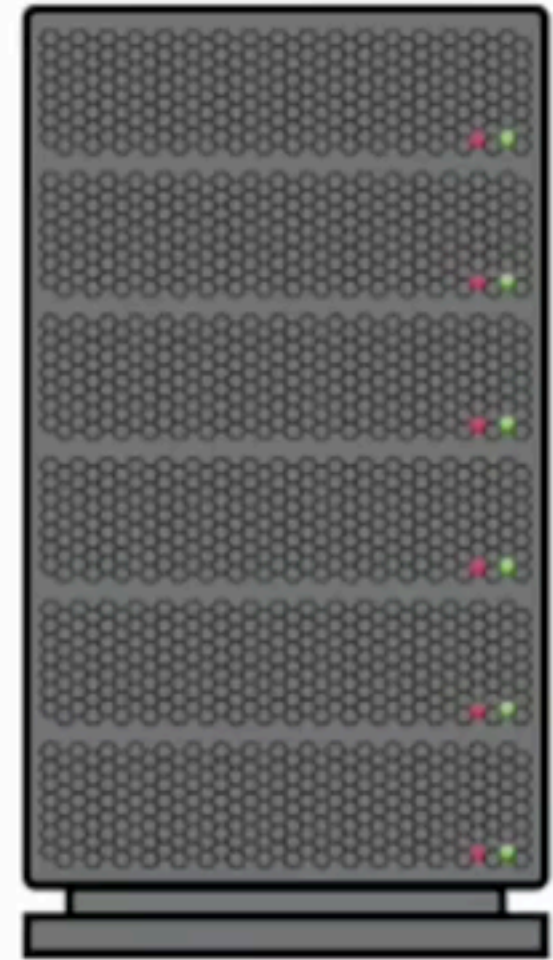
AJAX



HTTP

동기 통신 방식 Synchronous

Request \rightleftharpoons Response



AJAX

비 동기 통신 방식 ASynchronous

Request \rightleftharpoons Response

TEXT
HTML
XML
JSON

AJAX



Adaptive Path, CCO
Jesse James Garrett

AJAX 용어에 XML이 포함되어 있기 때문에 XML 만 통신 가능한 것이 아닐까 생각되겠지만...

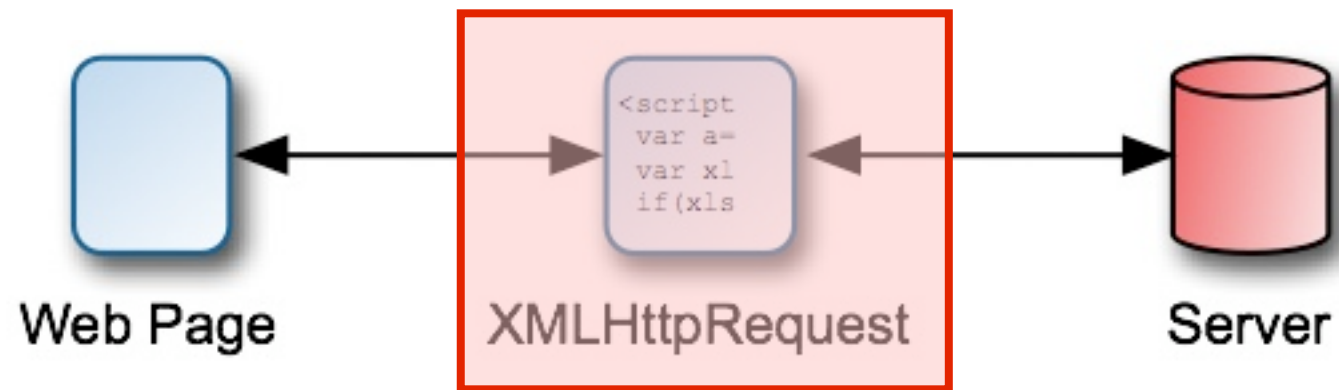
XML 뿐만 아니라,
TEXT, HTML, JSON 등도
처리가 가능합니다.

엑스 에이치 알 ?

XHR ?!

XML Http Request

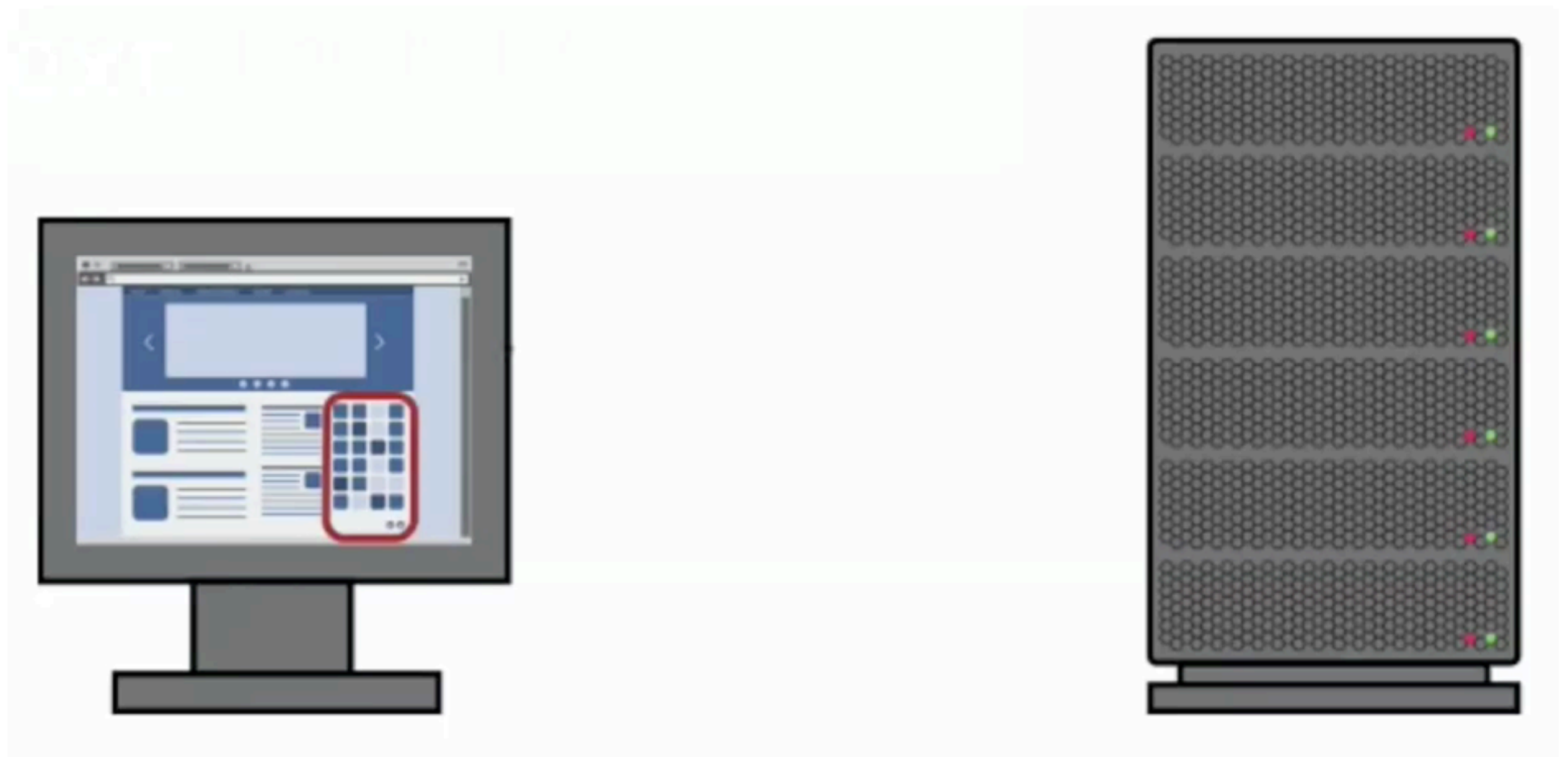
AJAX 비동기 통신을 하기 위한 객체



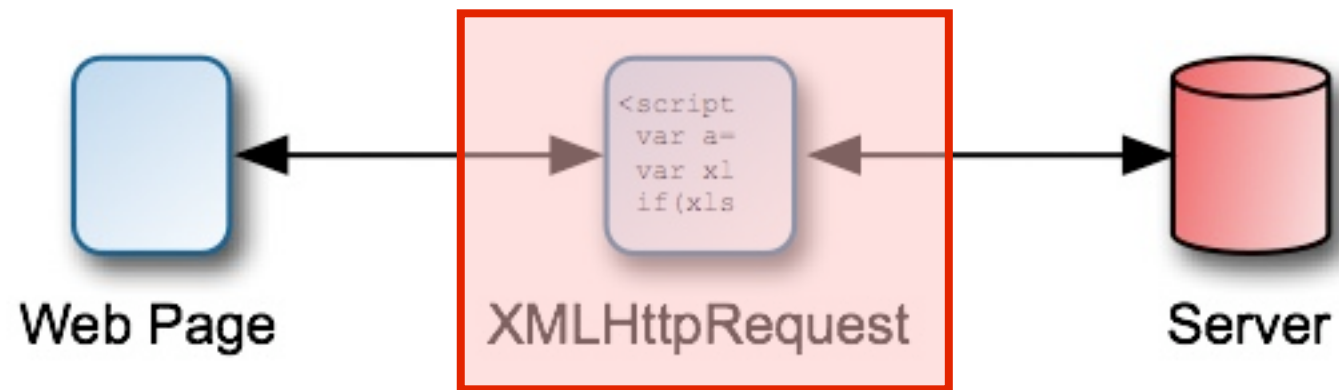
Clearleft
Jeremy Keith

AJAX 통신을 위해서는 **XHR 객체**를
사용해야 합니다.

XHR



AJAX need XHR



Clearleft
Jeremy Keith

XHR 객체는 생성자를 통해 생성할 수 있어요.
즉, `new`를 통해 생성해야 하죠.

`new XMLHttpRequest;`

```
> var xhr = new XMLHttpRequest;  
undefined  
> xhr  
▼ XMLHttpRequest {statusText: "", status: 0, response: "", responseType: "", responseXML: null...}  
  onabort: null  
  onerror: null  
  onload: null  
  onloadend: null  
  onloadstart: null  
  onprogress: null  
  onreadystatechange: null  
  readyState: 0  
  response: ""  
  responseText: ""  
  responseType: ""  
  responseXML: null  
  status: 0  
  statusText: ""  
  ▶ upload: XMLHttpRequestUpload  
    withCredentials: false  
  ▶ __proto__: XMLHttpRequest
```

Create XHR

```
▼ __proto__: XMLHttpRequest
  DONE: 4
  HEADERS_RECEIVED: 2
  LOADING: 3
  OPENED: 1
  UNSENT: 0
  ▶ abort: function abort() { [native code] }
  ▶ addEventListener: function addEventListener() { [native code] }
  ▶ constructor: function XMLHttpRequest() { [native code] }
  ▶ dispatchEvent: function dispatchEvent() { [native code] }
  ▶ getAllResponseHeaders: function getAllResponseHeaders() { [native code] }
  ▶ getResponseHeader: function getResponseHeader() { [native code] }
  ▶ open: function open() { [native code] }
  ▶ overrideMimeType: function overrideMimeType() { [native code] }
  ▶ removeEventListener: function removeEventListener() { [native code] }
  ▶ send: function send() { [native code] }
  ▶ setRequestHeader: function setRequestHeader() { [native code] }
  ▶ __proto__: Object
```

XHR Constructor


```
var xhr = new XMLHttpRequest;  
xhr.open('GET', 'ajax/data.txt', false);  
xhr.send();
```

통신 방법 : GET / POST

통신 파일 : HTML, XML, TEXT, JSON

비 동기 통신 설정 : true / false

Create & Open & Send

```
> var xhr = new XMLHttpRequest;  
undefined  
> xhr  
▼ XMLHttpRequest {statusText: "", status: 0, response: "", responseType: "", responseXML: null...}  
  onabort: null  
  onerror: null  
  onload: null  
  onloadend: null  
  onloadstart: null  
  onprogress: null  
  onreadystatechange: null  
  readyState: 0  
  response: ""  
  responseText: ""  
  responseType: ""  
  responseXML: null  
  status: 0  
  statusText: ""  
  ▶ upload: XMLHttpRequestUpload  
    withCredentials: false  
  ▶ __proto__: XMLHttpRequest
```

100 : continue
101 : Switching Protocols
200 : OK
404 : Client Error
5xx : Server Error

Check Status

```
var xhr = new XMLHttpRequest;  
xhr.open('GET', 'ajax/data.txt', false);  
xhr.send();  
  
if ( xhr.status === 200 ) {  
      
}  
}
```

Check Status

```
> var xhr = new XMLHttpRequest;  
undefined  
> xhr  
▼ XMLHttpRequest {statusText: "", status: 0, response: "", responseType: "", responseXML: null...}  
  onabort: null  
  onerror: null  
  onload: null  
  onloadend: null  
  onloadstart: null  
  onprogress: null  
  onreadystatechange: null  
  readyState: 0  
  response: ""  
  responseText: ""  
  responseType: ""  
  responseXML: null  
  status: 0  
  statusText: ""  
  ▶ upload: XMLHttpRequestUpload  
    withCredentials: false  
  ▶ __proto__: XMLHttpRequest
```

Receive Response

```
var xhr = new XMLHttpRequest;  
xhr.open('GET', 'ajax/data.txt', false);  
xhr.send();  
  
if ( xhr.status === 200 ) {  
    console.log(xhr, xhr.responseText);  
}
```

Receive Response

```
var xhr = new XMLHttpRequest;  
xhr.open('GET', 'ajax/data.txt', true);
```

```
xhr.send();
```

Setting Asynchronous

```
> var xhr = new XMLHttpRequest;  
undefined  
> xhr  
▼ XMLHttpRequest {statusText: "", status: 0, response: "", responseType: "", responseXML: null...}  
  onabort: null  
  onerror: null  
  onload: null  
  onloadend: null  
  onloadstart: null  
  onprogress: null  
  onreadystatechange: null  
  readyState: 0  
  response: ""  
  responseText: ""  
  responseType: ""  
  responseXML: null  
  status: 0  
  statusText: ""  
  ▶ upload: XMLHttpRequestUpload  
    withCredentials: false  
  ▶ __proto__: XMLHttpRequest
```

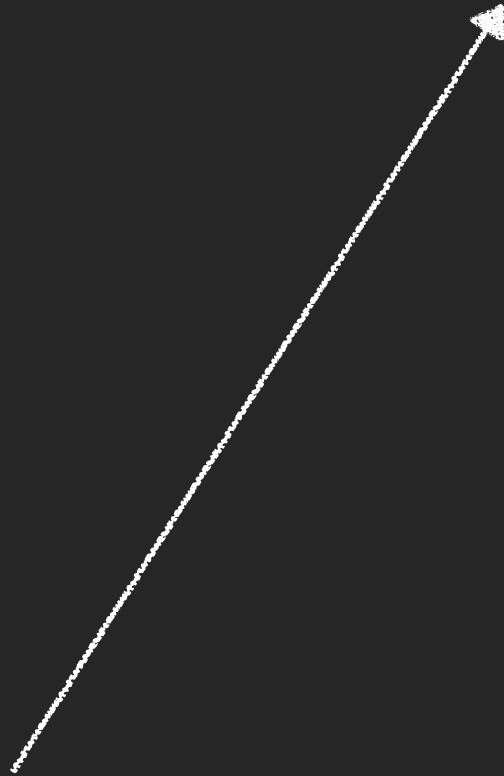
0: Uninitialized
1: Loading ...
2: Loaded
3: Interactive
4: Complete

Check Event & State


```
var xhr = new XMLHttpRequest;  
xhr.open('GET', 'ajax/data.txt', true);  
xhr.onreadystatechange = function() {  
    if ( xhr.status === 200 && xhr.readyState === 4) {  
        console.log(xhr, xhr.responseText);  
    }  
};  
xhr.send();
```

Check Event & State

브라우저 호환성 확보!



Cross Browsing



Clearleft
Jeremy Keith

XHR 객체를 IE 6 이하 버전은 지원하지 않아요.
대신 XMLHttpRequest 객체를 지원합니다.

ActiveXObject

```
function createXHR() {  
    return window.XMLHttpRequest ?  
        new XMLHttpRequest :  
        new ActiveXObject( 'Microsoft.XMLHTTP' );  
}
```

```
var xhr = createXHR();
```

createXHR

```
function createXHR() {  
    return window.XMLHttpRequest ?  
        new XMLHttpRequest :  
        new ActiveXObject('Microsoft.XMLHTTP');  
}  
  
var xhr = createXHR();  
xhr.open('GET', 'ajax/data.txt', true);  
xhr.onreadystatechange = function() {  
    if ( xhr.status === 200 && xhr.readyState === 4) {  
        console.log(xhr, xhr.responseText);  
    }  
};  
xhr.send();
```



XMLHttpRequest

XMLHttpRequest Redirect 1에서 리다이렉트됨

이력

편집

XMLHttpRequest는 Microsoft가 만든 [JavaScript](#) 개체(object)입니다. 후에 Mozilla도 이것을 받아들였습니다. XMLHttpRequest는 HTTP를 통해서 쉽게 데이터를 받을 수 있게 해줍니다. 이름과는 좀 동떨어지게도 XML 문서 이상의 용도로 쓰일 수 있습니다. Gecko에서 이 개체는 [nsIJSXMLHttpRequest](#)와 [nsIXMLHttpRequest](#) 인터페이스를 구현한 개체입니다. 최신 버전 Gecko에서 이 개체에 변경 사항이 좀 있었습니다. [XMLHttpRequest changes for Gecko1.8](#)을 보십시오.

기본 사용

XMLHttpRequest의 사용법은 아주 간단합니다. 이 개체의 인스턴스를 만들고, URL을 열고, 요청을 보내면 됩니다. 그 후에는 인스턴스의 결과 문서와 HTTP 상태 코드를 사용할 수 있게됩니다.

예

```
var req = new XMLHttpRequest();
req.open('GET', 'http://www.mozilla.org/', false);
req.send(null);
if(req.status == 200)
    dump(req.responseText);
```

참고: 이 예제는 동기적으로 동작하므로 이 함수를 JavaScript에서 호출하면 UI가 멈춥니다. 실제 제품 코드에서는 이 코드를 사용하지 마십시오.

비동기 사용

TABLE OF CONTENTS

기본 사용

예

비동기 사용

예

프로그레스 모니터링

다른 속성과 메소드

[responseXML](#)

[overrideMimeType\(\)](#)

[setRequestHeader\(\)](#)

[getResponseHeader\(\)](#)

[XPCOM에서 사용하기](#)

[References](#)

태그 파일