IERG4210 Web Programming and Security

Forms I - Client-side Implementations

Sherman Chow
Fall, 2021
Dept. of Information Engineering
The Chinese University of Hong Kong
Adapted from slides by Dr. Adonis Fung (and Prof. Kehuan Zhang)

1

Agenda

- HTTP
  - Introduction & Client-Server Model
  - HTTP Request and Response
- HTML Forms: Basics and Input Controls
- Client-side Restrictions
  - HTML: The use of form elements
  - HTML: HTML5 Validations
  - JS: Javascript Validations
- Form Submission Approaches
  - Traditional Form Submission
  - Programmatic Form Submission
  - AJAX Form Submission

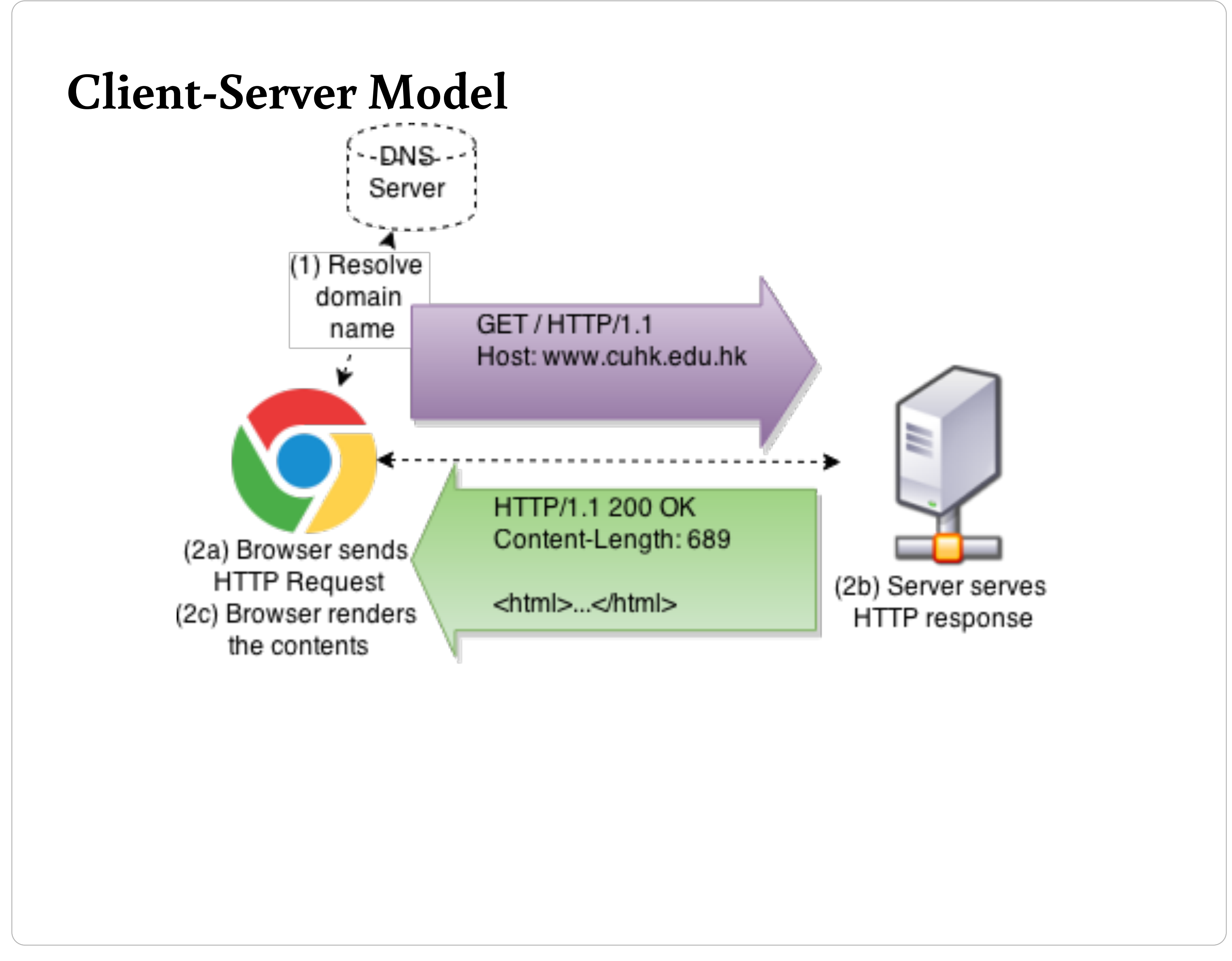
Introduction to HTTP

HTTP is a text-based application-layer protocol that defines how content is requested from a client application and served by a web server.

- Work on top of TCP/IP
- Latest standard is HTTP/2.0, defined in RFC7540 (HTTP/3 drafting)
- Specifications of HTTP Request and Response Headers

Client-Server Model

- Popular servers: Apache, Nginx, Node.js, IIS, AppEngine
- Popular clients/agents: Chrome, Firefox, IE, Safari
- (Demo) Using telnet to make a simple (text-based) request



Client-Server Model
DNS Server
(1) Resolve domain name
GET / HTTP/1.1
Host: www.cuhk.edu.hk
HTTP/1.1 200 OK
Content-Length: 689
<html>...</html>
(2a) Browser sends HTTP Request
(2c) Browser renders the contents
(2b) Server serves HTTP response

Surfing the Web using Telnet

```
$ telnet www.cuhk.edu.hk 80
Trying 137.189.99.27....
Connected to www.ie.cuhk.edu.hk.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.ie.cuhk.edu.hk

HTTP/1.1 301 Moved Permanently
Date: Wed, 03 Feb 2021 18:11:26 GMT
Server: Apache
Location: https://www.ie.cuhk.edu.hk/
Content-Length: 235
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a
href="https://www.ie.cuhk.edu.hk/">here</a>.</p>
</body></html>
```

Typical HTTP Requests

GET request:

```
GET
/~ierg4210/lectures/incl/process.php?q=abc HTTP/1.1
Host: course.ie.cuhk.edu.hk
```

POST request:

```
POST
/~ierg4210/lectures/incl/process.php?q=abc HTTP/1.1
Host: course.ie.cuhk.edu.hk
Content-Length: 105
Content-Type: application/x-www-form-urlencoded

name=Sherman&gender=M&email=xxx%40ie.cuhk.edu.hk&address=SHB%2C+CUHK%2C+NT&region=NT&action=
```

Specifications:

- Version: HTTP/1.0, HTTP/1.1, HTTP/2.0
- Method: GET, POST, PUT, HEAD, DELETE, TRACE, OPTIONS, CONNECT, etc...
- Parameters: query string vs. body
- Headers: hostname, content-length, content-type

Typical HTTP Response

```
HTTP/1.1 200 OK
Date: Mon, 26 Jan 2015 17:00:28 GMT
Content-Length: 413
Content-Type: text/html

<HTML>...</HTML>
```

Specifications:

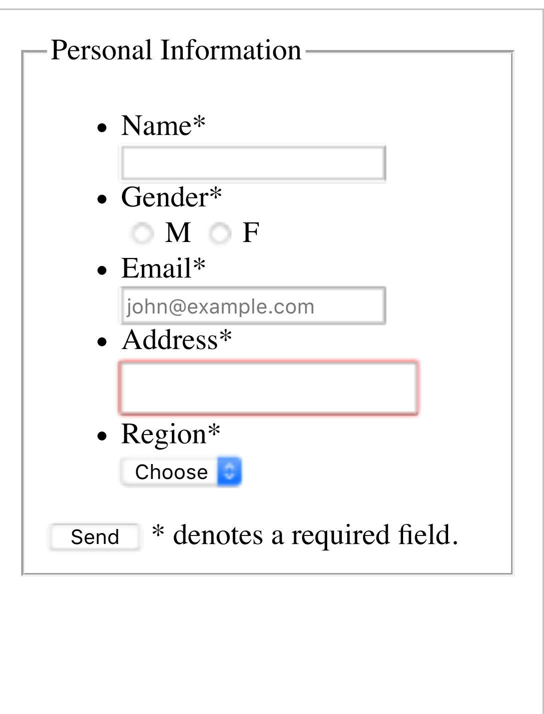
- Version: HTTP/1.0, HTTP/1.1, HTTP/2.0
- Status: 1xx for Informational, 2xx for Successful, 3xx for Redirection, 4xx for Client Error, and 5xx for Server Error
- Headers: content-length, content-type, and many more for authentication, cookies, security, caching, redirection
- Body: the content

HTML Forms

The prevalent approach to solicit information from users

A `<form>` tag that comprises different form controls, e.g., `<input>`, `<select>`, see a typical example below:

```
<fieldset>
 <legend>Personal Information</legend>
 <form method="POST" action="incl/process.php"><ul><li>
    <label>Name*</label>
    <div><input type="text" name="name" required /></div>
  </li><li>
    <label>Gender*</label>
    <div><input required type="radio" name="gender" value="M" /> M
<input type="radio" name="gender" value="F" /> F</div>
  </li><li>
    <label>Email*</label>
    <div><input type="email" name="email" required
placeholder="john@example.com" /></div>
  </li><li>
    <label>Address*</label>
    <div><textarea name="address"
required></textarea></div>
  </li><li>
    <label>Region*</label>
    <div><select required name="region">
<option value="">Choose</option>
<option value="HK">HK</option>
<option value="KL">KL</option>
<option value="NT">NT</select>
</div>
```



<form> Attributes

A typical `<form>` takes at least two attributes:

```
<form method="POST" action="process.php">
  <!-- included here are some form controls -->
</form>
```

`method="POST"` or `method="GET"` (default: GET)

- POST is used if a request is going to incur permanent change on server data; while GET is used for retrieving data

`action="process.php"` (default: the current URL)

- the value takes a URL that will accept the form request

`onsubmit="return false"` is optional

- Often used when the form is submitted over AJAX (to be discussed in later slides)

`enctype="multipart/form-data"` is optional

- When `<input type="file"/>` is used for file upload

Form Controls (1/4: Most Common Controls)

A typical form control is defined as follows:

```
<!-- <label> is to focus on a field when clicked -->
<label for="field1">Field 1: </label>
<input type="text" name="param1" id="field1" />
```

Field 1:

Text field

```
First Name: <input type="text" name="firstname"
value="Sherman"  />
```

First Name: Sherman

Password field (MUST use POST method)

```
Password: <input type="password" name="name"
value="abc" />
```

Password: •••

Hidden Field

```
Hidden? <input type="hidden" name="action"
value="updateData" />
```

Hidden?

Form Controls (2/4: Offering Choices)

Radio box
(limit to a single choice for radios under the same name)

```
<input type="radio" name="sex" value="M" checked="true"
/> Male
<input type="radio" name="sex" value="F" /> Female
```

Male Female

Checkboxes (multiple choices)

```
<input type="checkbox" name="item[]" value="A"
checked="true" /> A
<input type="checkbox" name="item[]" value="B" /> B
```

A B

Dropdown menu
(single selection; or try the multiple attribute)

```
Which OS do you like:
<select name="OS">
  <option name="1">iOS</option>
  <option name="2" selected="true">Android</option>
</select>
```

Which OS do you like: Android

Form Controls (3/4: More Controls)

Textarea (Multi-line text field)

```
Description:
<textarea name="desc" > text to be displayed
</textarea>
```

Description: text to be displayed

File Field

```
Photos: <input type="file" name="pics" />
```

Photos: Browse... No file selected.

Submit Button

```
<input type="submit" value="Go" />
```

Go

Submit Image Button (Image Credit: HSBC)

```
<input type="image" src="incl/go.gif" />
```

Go

Form Controls (4/4: *HTML 5 New Controls*)

Email and Date Field

```
<form>Email:* <input type="email" name="email" required
/> <input type="date" /> </form>
```

Email:*
mm / dd / yyyy

URL Field with optional use of styling by new CSS selectors

```
<style>:valid{border:1px solid #0F0}
       :invalid{border:1px solid #F00}</style>
<form>URL: <input type="URL" name="url" /></form>
```

URL:

Search Field

```
<form><input type="search" name="q"
placeholder="Search..." /></form>
```

Search...

Custom Pattern Field with regular expressions

```
<form>Amount: $<input type="text" name="amount"
pattern="^[\d,.]+$" /></form>
```

Amount: $

In a nutshell, HTML5 Forms introduced

- Tags with more semantic information: Built-in support of client-side validations
- New CSS pseudo-class: `:valid`, `:invalid`, `:required` and `:optional` (keyword for a selected element dependent on its content or external factors)

Regular Expressions

A language to recognize string patterns

Refer to a Cheatsheet for reference What you must know:

- ^ - start of string; $ - end of string (IMPORTANT to validations!)
- + - one or more times; ? - 0 or 1 times; * - 0 or more times

Examples:

Float (\d includes digits only):

```
^[\d\.]+$
```

Alphanumeric (\w includes letters, digits, underscore):

```
^[\w\-, ]+$
```

Email (apparently '\' is the *escape character* here):

```
^[\w\-\/][\w\'\-\/\.]*@[\w\-]+(\.[\w\-]+)*(\.[\w]{2,6})$
```

The regular expression for email address is readily available on Web.
(You need to first know what constitutes a valid email address to write this regex.)
IMPORTANT: Consult *credible* websites for reusable *rigorous* patterns!!

https://regexr.com

HTML5 Forms: Browser Support - Types

	Firefox	Safari	Safari	Chrome	Opera	IE	Android
Email	4+	5+	3.1+	6+/10+	10.6+	10+	4+
Tel	4+	5+	3.1+	6+	10.6+	10+	2.3+
Url	4+	5+	3.1+	6+/10+	10.6+	10+	2.3+
Search	4+	5+	4+	6+	10.6+	9/10+	4+
Color	29+	8+	8-	20+	11+	11-	4.4+
Number	29+	5+	3.2+	7+	9+	10+	2.3+
Range	23+	4+	5+	6+	11+	10+	4.2+
Date	32-	7-	5+	20+	9+	11+	4.4+
Text	All	All	All	All	All	All	All

A partial list captured from https://wufoo.com/html5/#types

HTML5 Forms: Browser Support - Attributes

	Firefox	Safari	Safari	Chrome	Opera	IE	Android
Placeholder	4+	4+	4+	10+	11.10+	10+	2.3+
Autofocus	4+	5+	5-	6+	11+	10+	3+
Maxlength	4.4+	5+	4+	6+	11+	9+/10	2.3+
List (Datalist)	4+	7-	7-	20+	9+	10+	4.3-
Autocomplete	4+	5.2+	6+	14+	10.6+	11+	4.4+
Required	6+	5+	4+	6+	10.6+	10+	2.3+
Pattern	4+	5+	4+	10+	11+	10+	2.3+
Spellcheck	3.6+	4+	7+	10+	11+	10+	4.3-
Novalidate	4+	7-	7-	10+	10+	10+	4.2-

A partial list captured from https://wufoo.com/html5/#attributes

Client-side Restrictions

- To inform the users early on for input errors
  - To create a more interactive and responsive UI experience
  - Otherwise, input errors are prompted only after form submissions (round-trip delay)
- To imply a specific pattern that a user is expected to follow
  - To help users enter/choose the valid data that we need
  - Yet, these restrictions can be bypassed by Parameter Tampering Attacks!! Don't count on them for security!!
  - Reason: A user has full control of any client-side code downloaded to his browser using the lovely Firebug :)
- Hence, you need input validations implemented on BOTH
  - *server-side* for *security enforcement*, and
  - *client-side* for better *user experience*.

3 Approaches of Client-side Restrictions

1. The use of different form controls
   - e.g., Radioboxes for genders implies a restriction on only two possible values, i.e., M or F
   - e.g., Dropdown menu implies a restriction on accepting some default choices
2. Validations with HTML5 (shown in the previous slide)
   - The first built-in support of client-side validations by IE 10+, Firefox 4+, Chrome, etc
   - e.g., Email, URL, Search, and Custom fields we just see
3. Validations with Javascript (to be discussed in next slide)
   - The programmatic way to customize input patterns
   - Well-supported across browsers

Form Validations with Javascript (1/4)

Strategy: Write your code in HTML5 for new browsers; Fallback to Javascript for legacy ones

Given a form that has an HTML5 Email field,

```
<form id="loginForm" method="POST">
  Email: <input type="email" name="em" /><br/>
  Password: <input type="password" name="pw" /><br/>
  <input type="submit" value="Login" />
</form>
```

Note: Unsupported *type* will fallback to an ordinary textfield

Add the title, HTML5 requried and pattern attributes

```
<form id="loginForm" method="POST">
  Email: <input type="email" name="em" title="valid email" required
      pattern="^[\w\-\/][\w\'\-\/\.]*@[\w\-]+(\.[\w\-]+)*(\.[\w]{2,6})$" /><br/>
  Password: <input type="password" name="pw"
      title="valid password" required /><br/>
  <input type="submit" value="Login" />
</form>
```

Note:

- Unsupported *attributes* will be ignored in legacy browsers
- The regular expression for email address is readily available on Web.

Form Validations with Javascript (2/4)

To validate the form right before form submission:

```
<form id="loginForm" method="POST">
  Email: <input type="email" name="em" title="valid email" required
      pattern="^[\w\-\/][\w\'\-\/\.]*@[\w\-]+(\.[\w\-]+)*(\.[\w]{2,6})$" /><br/>
  Password: <input type="password" name="pw"
      title="valid password" required /><br/>
  <input type="submit" value="Login" />
</form>
<script type="text/javascript">
var loginForm = document.getElementById('loginForm');
// Do this only if the HTML5 Form Validation is absent
if (!loginForm.checkValidity || loginForm.noValidate)
  // to listen on the submit event of "loginForm"
  loginForm.onsubmit = function(){
    // a private function for displayError
    function displayErr(el,msg){alert('FieldError: ' + msg);el.focus();return false}
    // looping over the array of elements contained in the form
    for (var i = 0, p, el, els = this.elements; el = els[i]; i++) {
      // validate empty field if required attribute is present
      if (el.hasAttribute('required') && el.value == '')
        return displayErr(el, el.title + ' is required');
      // validate pattern if pattern attribute is present
      if ((p = el.getAttribute('pattern')) && !new RegExp(p).test(el.value))
        return displayErr(el, 'in' + el.title);
    }
    // If false is returned above, the form submission will be canceled;
    // If false is NOT returned, the form will submit accordingly
  }
</script>
```

Form Validations with Javascript (3/4)

With an HTML5-compilant browser, JS validation is ignored:

Email:
Password:
Login

Note: POST Parameters can be accessed only by server but not JS. Hence, nothing is shown here after submission. Firebug can show what was already sent.

With HTML5 Validation disabled w/`novalidate` attribute:

Using `<form novalidate>`

Email:
Password:

Note: Need some free old-school IE browsers for professional compatibility tests!?

Form Validations with Javascript (4/4)

Recall the best practice: Graceful Degradation (in Lecture 2)

- if (HTML5 supported) use the native HTML5 Validation
- else if (JS supported) use the JS validation code
- else the form still works without any validations

Extend the code to also validate `radio` and `checkbox`

```
for (var i = 0, p, el, els = this.elements; el = els[i]; i++) {
  // validate empty field, radio and checkboxes
  if (el.hasAttribute('required')) {
    if (el.type == 'radio') {
      if (lastEl && lastEl == el.name) continue;
      for (var j = 0, chk = false, lastEl = el.name, choices = this[lastEl],
           choice; choice = choices[j]; j++)
        if (choice.checked) {chk = true; break;}
      if (!chk) return displayErr(el, 'choose a ' + el.title);
      continue;
    } else if ((el.type == 'checkbox' && !el.checked) || el.value == '')
      return displayErr(el, el.title + ' is required');
  }
  if ((p = el.getAttribute('pattern')) && !new RegExp(p).test(el.value))
    return displayErr(el, 'in' + el.title);
}
```

Code Demo. A question: how to skip disabled/hidden controls??

3 Form Submission Approaches

1. **Traditional Form Submission** (demonstrated in the previous slide)
   - Triggered by a submit button *or* the Enter key
   - Fires the `submit` event, where one can validate before a form submission
2. **Programmatic Form Submission**
   - Recommended to use this only when submiting a form automatically

```
<form method="POST" id="buildAutoPostReq"><!-- Some hidden fields here --></form>
<script type="text/javascript">document.forms[0].submit();</script>
```

   - Unfortunately, programmers (incl. HSBC) who don't know `<input type="image">` like to do this for images: When an image is clicked, `Form.submit()` will be finally called if a form is properly validated
   - BAD: NO `submit` event is fired. Without code analysis, difficult to know whether a submission has actually occurred
3. **AJAX Form Submission** (to be discussed in the next slide)
   - AJAX: *Asynchronous Javascript and XML*; It's all about the `XMLHttpRequest` API, study it before using it to submit form data

AJAX Form Submission (1/3)

Demonstration:

Email:
Password:
Login

Feedback from Server:

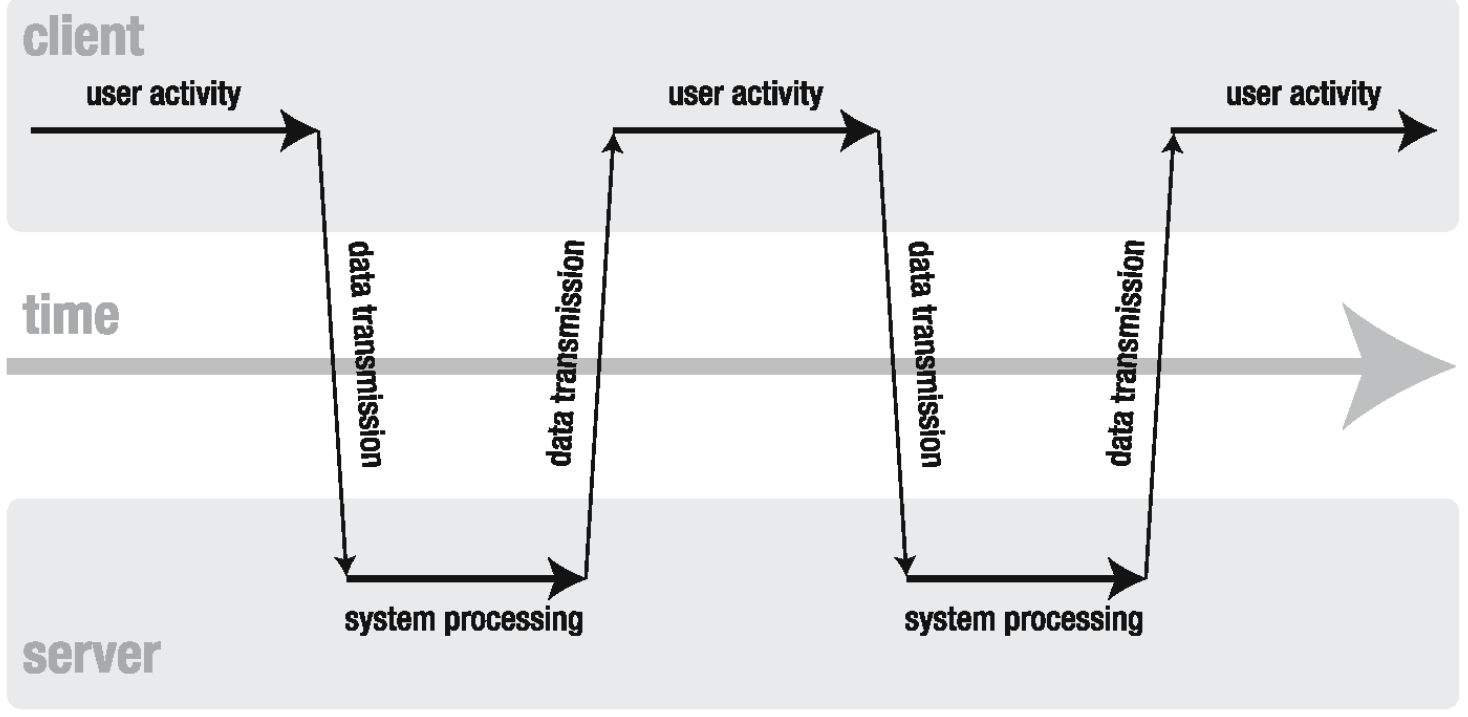
Nothing yet

(You can check Headers, Request, and Response of the "Network" tab in the Developer Tool)

Advantages:

- Modern user experience
  - Eliminate page-load effect (no blank screen);
  - Only load the changed part when it "arrives"
- Using the well-known `XMLHttpRequest` API
  - Sends requests at background; not limited to only send form data :)
- Cancel the default form submissions
  - returns `false` in the `submit` event

AJAX: Synchronous vs. Asynchronous (1/3)



As opposed to asynchronous calls, synchronous calls are blocking (hangs) until the server returns, i.e., less efficient

Image Source: John Resig, Pro Javascript Techniques, p.26, 2007

AJAX: Synchronous vs. Asynchronous (2/3)

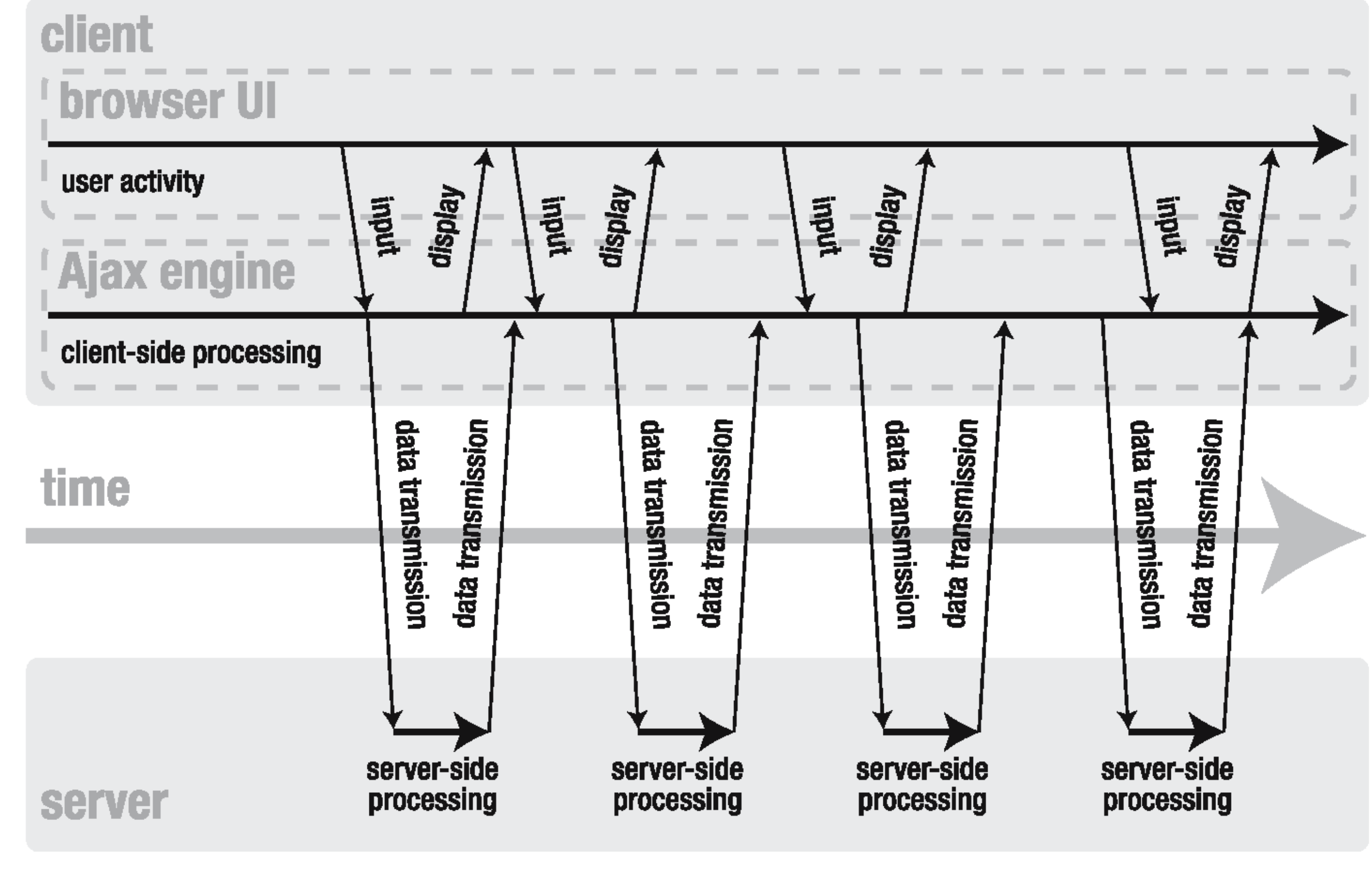


Image Source: John Resig, Pro Javascript Techniques, p.26, 2007

AJAX: Synchronous vs. Asynchronous (3/3)

Principle: Do something else while eating

Dispatch many requests at a time. Also do something else.
Get notified when the server returns, then render the results.
The responses will likely be out of order.

Typical workflow in AJAX Form submission (shown in the previous slide)

1. Listen to `submit` event
2. Cancel the default form submission
3. Craft a `POST` request to send over AJAX
4. On feedback received, echo the feedback

AJAX: Implementation w/ `XMLHttpRequest`

```
// e.g., to call, myLib.ajax({url:'process.php?q=hello',success:function(m){alert(m)}});
  myLib.ajax = function(opt) { opt = opt || {};
    var xhr = (window.XMLHttpRequest                      // Usu. ?/|| is for compatibility
            ? new XMLHttpRequest()                        // IE7+, Firefox1+, Chrome1+, etc
            : new ActiveXObject("Microsoft.XMLHTTP"),     // IE 6
        async = opt.async || true,
        success = opt.success || null, error = opt.error || function(){/*displayErr()*/};
    // pass three parameters, otherwise the default ones, to xhr.open()
    xhr.open(opt.method || 'GET', opt.url || '', async); // 3rd param true = async
    if (opt.method == 'POST')
      xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    // Asyhronous Call requires a callback function listening on readystatechange
    if (async)
      xhr.onreadystatechange = function(){
        if (xhr.readyState == 4) { // 4 is "more ready" than 3 or 2
                    var status = xhr.status;
          if ((status >= 200 && status < 300) || status == 304 || status == 1223)
            success && success.call(xhr, xhr.responseText); // raw content of the response
          else if (status < 200 || status >= 400)
            error.call(xhr);
        }
      };
    xhr.onerror = function(){error.call(xhr)};
    // POST parameters encoded as opt.data is passed here to xhr.send()
    xhr.send(opt.data || null);
    // Synchronous Call blocks UI and returns result immediately after xhr.send()
    !async && success && success.call(xhr, xhr.responseText);
  };
```

Ref: https://developer.mozilla.org/en/AJAX/Getting_Started

AJAX Form Submission (2/3)

To generate POST parameters based on the control values

```
myLib.formData = function(form) {
  // private variable for storing parameters
  this.data = [];
  for (var i = 0, j = 0, name, el, els = form.elements; el = els[i]; i++) {
    // skip those useless elements
    if (el.disabled || el.name == ''
        || ((el.type == 'radio' || el.type == 'checkbox') && !el.checked))
      continue;
    // add those useful to the data array
    this.append(el.name, el.value);
  }
};
// public methods of myLib.formData
myLib.formData.prototype = {
  // output the required final POST parameters, e.g., a=1&b=2&c=3
  toString: function(){
    return this.data.join('&');
  },
  // encode the data with the built-in function encodeURIComponent
  append: function(key, val){
    this.data.push(encodeURIComponent(key) + '=' + encodeURIComponent(val));
  }
};
```

So, this can feed the data parameter for `myLib.ajax({data:""})`

Note: you may refer to last week's lecture for String Concatenation

AJAX Form Submission (3/3)

We build another reusable function submitOverAJAX()

```
myLib.submitOverAJAX = function(form, opt) {
  var formData = new myLib.formData(form);
  formData.append('rnd', new Date().getTime());
       opt = opt || {};
       opt.url = opt.url || form.getAttribute('action');
       opt.method = opt.method || 'POST';
       opt.data = formData.toString();
       opt.success = opt.success || function(msg){alert(msg)};
  myLib.ajax(opt);
};
```

Finally, specify the form and a corresponding callback

```
function el(A) {return document.getElementById(A)};

var loginForm = el('loginForm');
loginForm.onsubmit = function(){
  // submit the form over AJAX if it is properly validated
  myLib.validate(this) && myLib.submitOverAJAX(this, {success:function(msg){
       el('result').innerHTML = 'Echo from Server: $_POST = ' + msg.escapeHTML();
  }});
  return false;     // always return false to cancel the default submission
}
```

Can we listen to the `click` event of the `submit` button instead? Why not?

Complicated? this final block is all you need to know (to call them) in assignment. :)

Our myLib.js so far...

When all the functions (incl. myLib.validate()) are built in a single library

```
(function(){
        String.prototype.escapeHTML = function(){
                return this.replace(/&/g,'&amp;').replace(/</g,'&lt;').replace(/>/g,'&gt;');
        }


  var myLib = window.myLib = (window.myLib || {});

  // To generate POST parameters based on the control values
  myLib.formData = function(form) {
    // private variable for storing parameters
    this.data = [];
    for (var i = 0, j = 0, name, el, els = form.elements; el = els[i]; i++) {
      // skip those useless elements
      if (el.disabled || el.name == ''
         || ((el.type == 'radio' || el.type == 'checkbox') && !el.checked))
        continue;
      // add those useful to the data array
      this.append(el.name, el.value);
    }
  };
  // public methods of myLib.formData
  myLib.formData.prototype = {
    // output the required final POST parameters, e.g. a=1&b=2&c=3
    toString: function(){
      return this.data.join('&');
    },
    // encode the data with the built-in function encodeURIComponent
    append: function(key, val){
      this.data.push(encodeURIComponent(key) + '=' + encodeURIComponent(val));
    }
  };

  myLib.ajax = function(opt) {
    opt = opt || {};
    var xhr = (window.XMLHttpRequest)
          ? new XMLHttpRequest()                     // IE7+, Firefox1+, Chrome1+, etc
          : new ActiveXObject("Microsoft.XMLHTTP"),  // IE 6
```

More on XMLHttpRequest

Using XMLHttpRequest

In this guide, we'll take a look at how to use `XMLHttpRequest` to issue HTTP requests in order to exchange data between the web site and a server. Examples of both common and more obscure use cases for `XMLHttpRequest` are included.

To send an HTTP request, create an `XMLHttpRequest` object, open a URL, and send the request. After the transaction completes, the object will contain useful information such as the response body and the HTTP status of the result.

```
function reqListener () {
  console.log(this.responseText);
}
```

https://developer.mozilla.org/en-US/docs/DOM/XMLHttpRequest/Using_XMLHttpRequest