ICT286
Web and Mobile Computing

Topic 9
Designing Single Page Applications with Ajax

Objectives

- Understand the problems of traditional web application, and the benefits of single page applications (SPA)
- Understand the requirements of a SPA and be able to design simple SPAs.
- Understand how Ajax is used to implement Single Page Applications
- Be able to handle Ajax plain text response.
- Be able to handle Ajax XML response returned from the server.
- Understand and be able to implement simple client server application using Ajax and PHP.
- Understand the JSON notation;

Readings

Sebesta: Ch 10.

Traditional Web Applications – Frequent Page Loading

- A traditional web application often consists of multiple pages. Moving between pages requires loading and reloading of the same and/or different pages
  - Result in low responsive applications due to time taken to load a new page and to render it
  - The user cannot interact with the application while it is loading a page – bad user experience
  - The user loses focus and flow as the result of switching between different pages
  - Excessive page reloading places unnecessary burden on the network

Traditional Web Applications – Retrieving Server Results

- In a traditional web application, most page re-loadings are caused by retrieving server results
  - Usually only small changes are needed on the existing page in order to incorporate the new results, eg, showing the total price
  - Nevertheless, a new page is loaded, and the entire existing page is replaced by the new page which is often very similar to the existing page
  - This is a great waste of resources (eg, human waiting time, processor time on both the client and server, and network bandwidth), resulting in irresponsive web application, and bad user experience.

Traditional Web Applications – Heavy Server Processing

- A traditional web application relies heavily on the server to provide most of processing
  - This includes many operations that could be performed on the client side, eg, generating HTML code using the results of the database query.
  - This places unnecessary burden on the server which needs to serve many clients.
  - This is another reason why a traditional web application is not as responsive as a desktop application.

Traditional Web Applications – Not Good for Offline Use

- As a traditional web application consists of many pages (either static or dynamic), all pages are not loaded at the beginning.
  - This is not major issue if the client runs on a desktop with a constant and reliable network connection (apart from slowness and high latency). The application only loads a new page when it is required.
  - But it becomes a problem if the client runs on a mobile device, because the mobile device often move in and out of service area. When the device is out of service area, the application cannot load a required page – the application stops functioning!

Single Page Applications

- Due to the problems experienced with the traditional web applications, a new way of designing web applications has come into vogue:

  Single Page Applications or SPA

  - The idea is that web applications should be as responsive as desktop applications
  - And they should be good for online use as well as for offline use. When the device is out of service area, the web application should continue to function by relying on the cached copy of code on the client device.
  - The application still retains some of the familiar behaviours of a traditional web application such as navigation and bookmarking.

Single Page Applications

- As the name implies, most SPA consists of a single HTML page that is loaded at the beginning and never need to be reloaded (unless for update).
  - Transition between different “pages” in the application actually corresponds to switching between different “views” implemented in the same HTML page
  - No reloading of a new page is required
  - No parsing of a new page is needed as the page has already been parsed, and its DOM is in the memory.
  - At any time, only one view is displayed, other views are hidden away from the user. Rendering of a new “view” takes very little time.
  - The user may continue to work on the application when the network connection is lost.

Single Page Applications

- Like a traditional web application, an SPA also needs to communicate with its server to obtain its service, such as to retrieve the details of a product, or to calculate the total price.
  - However in cases like this, only the data, rather than a new HTML page, are sent to the client.
  - The data sent to the client by the server are small in size compared to a new HTML page incorporating those data.
  - Once the client receives the data, some small areas of the existing page may be revised by manipulating the DOM, eg, a new price figure replacing the old price figure.

Single Page Applications

- More importantly, when an SPA requests data from the server, the operation is performed *asynchronously*
  - The user can continue to work on the web application while data are being retrieved *on the background*, the user experiences no pause in the application.
  - This is in sharp contrast with the traditional web application – the user is forced to wait until the new page is retrieved from the server, and is parsed, and rendered in the web browser. The application is effectively frozen before the new page is rendered on the screen. There is nothing the user can do on the application before the new page is displayed!

Single Page Applications

- For small to medium sized applications, an SPA typically consists of a single page that provides multiple views.
  – Once loaded, it does not need to be reloaded again.
- For large applications, the SPA may consists of more than one page.
  – Some of the pages may not be loaded initially
  – A page is loaded the first time it is needed
  – However, the page is only loaded once. Unlike in a traditional web application, the page will not be loaded again.

Single Page Applications

- Summary of requirements of a SPA:
  - It usually consists of a single page (particularly for small to medium sized applications);
  - Each page is only loaded once, and is never reloaded later;
  - The application provides multiple views, akin to multiple pages in a traditional web application;
  - Updating the current page is performed with asynchronous Ajax requests;
  - The application retains the familiar behaviours of a traditional web application, eg, the user can switch between different "pages", and can go back to previous pages, and can bookmark any of the "pages".

Single Page Applications

- Now there are excellent support for SPA. Many JavaScript libraries or frameworks are available that make SPA design much easier, eg
  - jQuery for DOM manipulation
  - AngularJS providing a framework for SPA
- In this topic, we will first discuss how to design a SPA with HTML, CSS and JavaScript through an example: Example 1
- After that we will focus on client-server data exchanges using Ajax and PHP
  - including using Ajax directly and through jQuery
  - we will also cover the use of JSON as the data format for client-server data exchanges.

Example 1: An Simple SPA

- The first requirement of an SPA is to provide multiple views in a single HTML document
- There are many ways to provide multiple views. One way is to provide several structured elements with the same visual outlooks
  - one of which is visible
  - and all others are hidden, eg,

```
<article id="page2" hidden="hidden">
    <p style="color: red"> This is page 2</p>
    <p style="color: red"> This is page 2</p>
</article>
```

Multiple Views

```
<!DOCTYPE html>
<html>
<head>
    <title>A rudimentary SPA</title>

    <meta name="viewport" content="width=device-width,
initial-scale=1.0"/>

    <link href="css/index.css" rel="stylesheet"/>
    <script src="js/jquery-3.3.1.js"></script>
    <script src="js/index.js"></script>
</head>
```

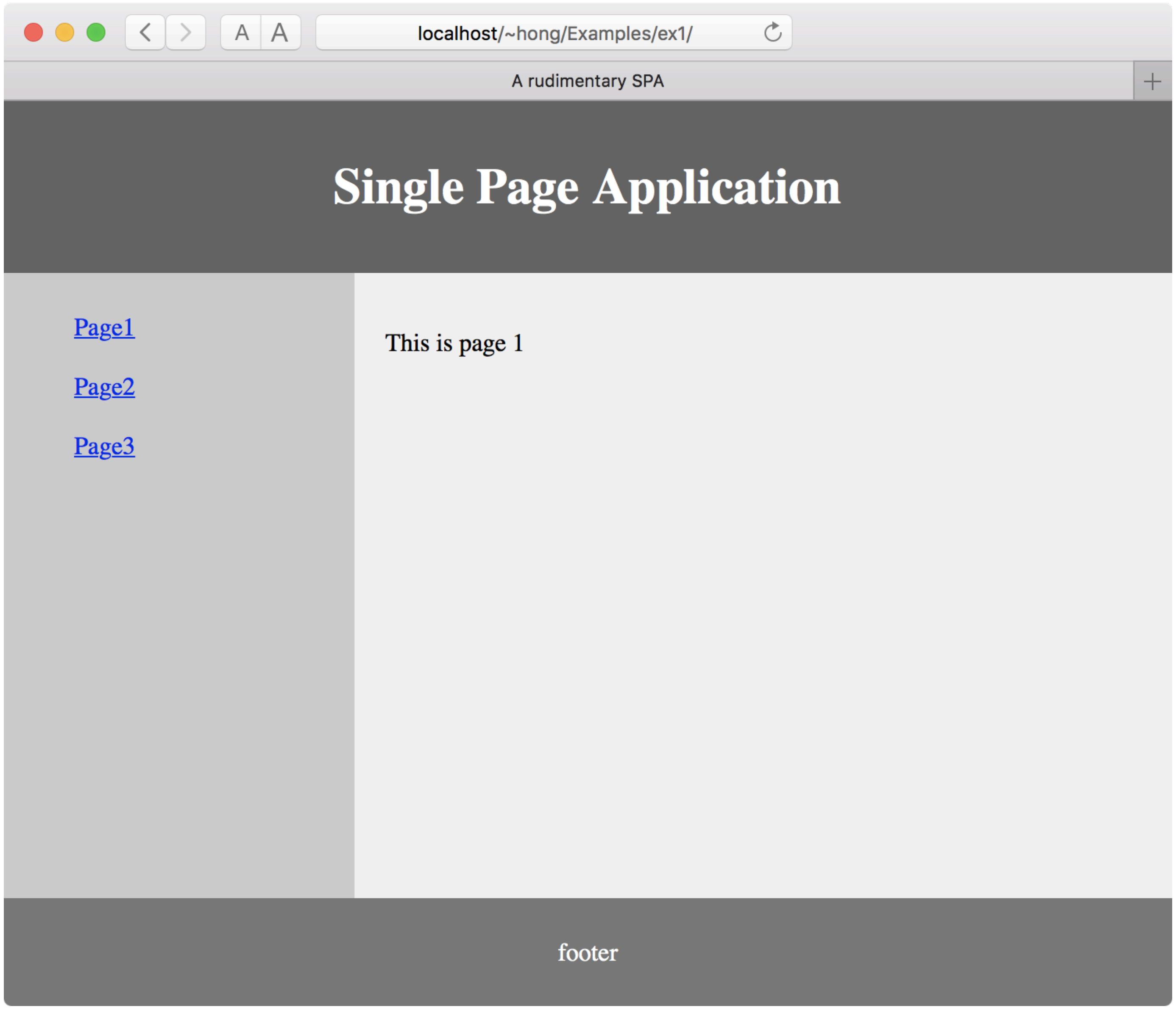
Multiple Views

```
<body>
    <header> <h1>Single Page Application</h1> </header>
    <div id="main">
        <nav>
            <ul>
                <li><a href="page1">Page1</a></li>
                <li><a href="page2">Page2</a></li>
                <li><a href="page3">Page3</a></li>
            </ul>
        </nav>
        <article id="page1">
            <p> This is page 1</p>
        </article>
```

Multiple Views

```
        <article id="page2" hidden="hidden">
            <p style="color: red"> This is page 2</p>
            <p style="color: red"> This is page 2</p>
        </article>
        <article id="page3" hidden="hidden"">
            <p style="color: green"> This is page 3</p>
            <p style="color: green"> This is page 3</p>
            <p style="color: green"> This is page 3</p>
        </article>
    </div>
    <footer><p>footer</p></footer>
</body>
```

Multiple Views



Single Page Application

Page1

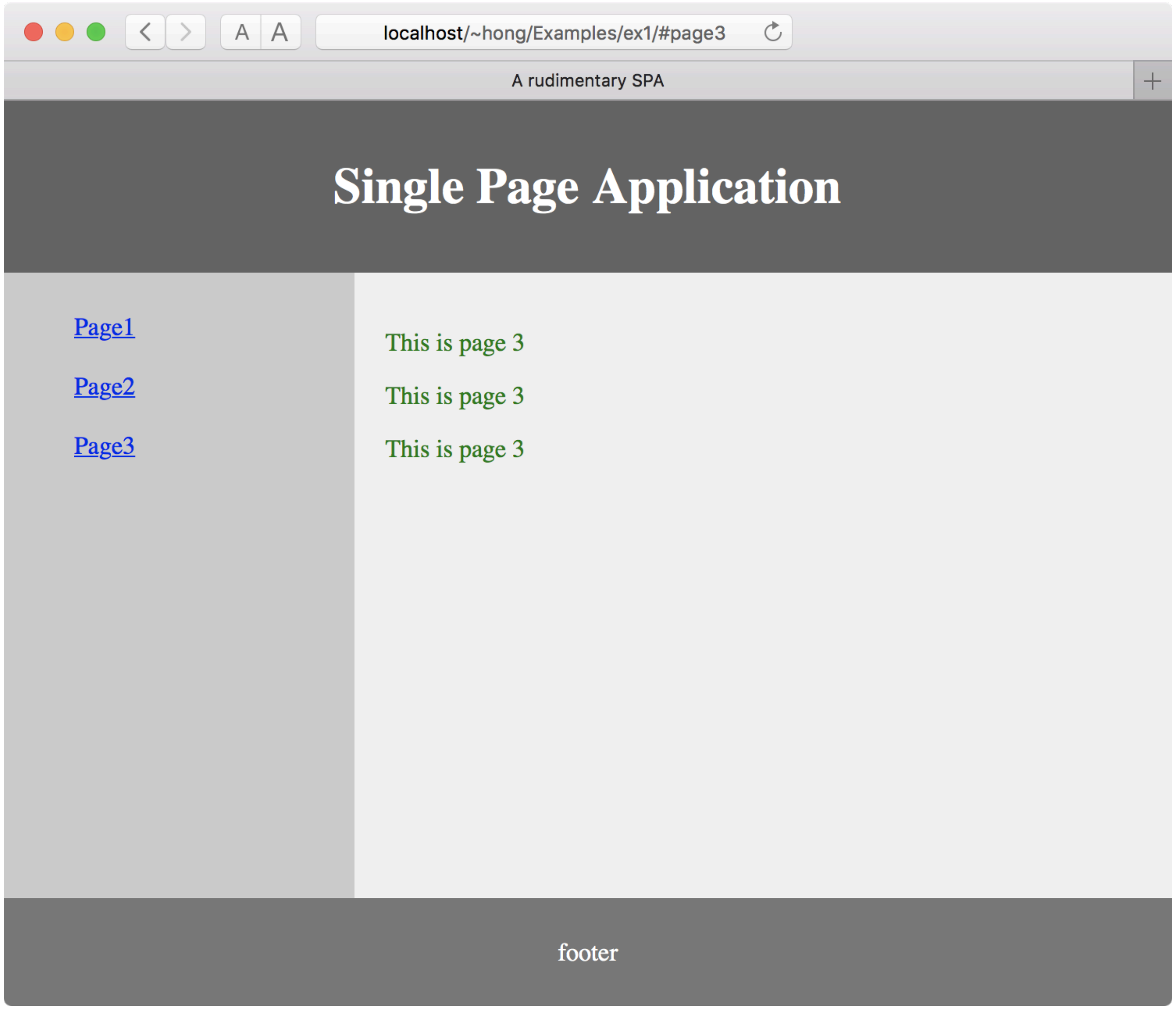
Page2

Page3

This is page 1

footer

Multiple Views



localhost/~hong/Examples/ex1/#page3

A rudimentary SPA

Single Page Application

Page1

Page2

Page3

This is page 3

This is page 3

This is page 3

footer

Switching Views

- The second requirement of an SPA is to support navigation to different views without causing reloading of the same page.
- One way to do it is to have navigation links pointing to different "pages". Each of these links represent one view and it is not a link to an HTML file on the server.

```
<nav>
    <ul>
        <li><a href="page1">Page1</a></li>
        <li><a href="page2">Page2</a></li>
        <li><a href="page3">Page3</a></li>
    </ul>
</nav>
```

Switching Views

- When a link is clicked, we need to
  - switch to the view represented by the link
  - and make sure the click does not cause the browser to attempt to re-load the url.
- This is achieved by registering an event handler for the click event on the anchor elements.

```
var page = ["#page1", "#page2", "#page3"];
var curPage = page[0];
   . . .
   $('nav a').click(function(e){
       e.preventDefault();
       var newPage = $(this).attr('href');
       window.location.hash=newPage;
   });
```

Switching Views

- In this example, we have three views. They are represented by three location hash values: `#page1`, `#page2` and `#page3`.
- When the user clicks one of the navigation link, the event handler will do the following:
  1) call `e.preventDefault()` to tell the browser not to load the url (the default behaviour is to load the url)
  2) find out the value of the href attribute of the link which is one of "`page1`", "`page2`" and "`page3`".
  3) change the hash value of the current address (`window.location.hash`) – this will trigger the "`hashchange`" event if the new hash value is different from the existing hash value.

Switching Views

- The following code registering the event handler for hashchange event:

```
$(window).on('hashchange', function(){
    var newPage = getPage(window.location.hash);
    render(newPage);
});
```

- When "`hashchange`" event is triggered, it will
  - get the hash value of the current address by calling

    ```
    getPage(window.location.hash)
    ```

  - then render the view by hiding the current view and showing the new view using

    ```
    render(newPage);
    ```

Switching Views

- The function `getPage(hash)` tries to compare the new hash value with the three hash values in the array `page`.
- If the new hash is one of array elements, it will return the new hash.
- Otherwise it will return `#page1` (eg, if the hash is empty or other strings)

```
var page = ["#page1", "#page2", "#page3"];
var curPage = pages[0];
function getPage(hash){
   var i = page.indexOf(window.location.hash);
   if (i<0 && hash!="")window.location.hash=page[0];
   return i < 1 ? page[0] : page[i];
}
```

Switching Views

- The function `render(newPage)` compares the new hash value with the hash value of the view currently in display.
- If the two are the same, no change to view.
- Otherwise hide the current view and show the new view.

```
function render(newPage){
    if (newPage == curPage) return;
    $(curPage).hide();
    $(newPage).show();
    curPage = newPage;
}
```

- Note `curPage` or `newPage` happens to be a selector for one of the three article elements, eg, if `curPage` is `"#page2"`, then `$("#page2")` is the jQuery object for the article element with `id="page2"`.

Switching Views

- The full example of this simple SPA is available in directory `Examples/ex1`.
- It includes three files:
  - `ex1/index.html`
  - `ex1/css/index.css`
  - `ex1/js/index.js`
- Note, when the application is first loaded, we call render function to render the view based on the hash value in the address.
- If there is no hash value in the address, we will render `#page1`.

Ajax

- Ajax stands for ***A****synchronous* ***J****avaScipt* ***a****nd* ***X****ML.*
- Ajax is a client-side technology that is used to retrieve data from the web server *asynchronously.*
- Ajax provides Web-based applications with responsiveness approaching that of desktop applications.
- Ajax is not a new language. It is a new way of writing web applications using existing protocols and languages: HTTP, JavaScript, HTML, JSON, XML, DOM and CSS.
- Ajax can be used together with many server-side technologies such as PHP, servlets and ASP.NET.
- The response from the server can be in the form of plain text, HTML, XML or JSON notation. Most applications use JSON for data exchange.

History of Ajax

- Possibility began with the `iframe` element, which appeared in IE4 and Netscape 4
  - An `iframe` element could be made invisible and could be used to send asynchronous requests
  - In 1999, Microsoft introduced `XmlDocument` and `XMLHTML` ActiveX objects in IE5 for asynchronous requests
  - A similar object is now supported by all current browsers
- Two events ignited widespread interest in Ajax:
  - The appearance of Google Maps and Google Mail around 2004
  - Jesse James Garrett named the technology "Ajax" in 2005

Main Differences

- Non-Ajax:
  - Each client request (as the result of clicking a link or submitting a form) results in downloading a new page from the server, replacing the current page with the new page and rendering the new page by the browser.
    - This happens even if only a tiny part of the current page needs update.
  - Slow: because the whole page needs to be retrieved from the server and the whole page needs be parsed and rendered.
  - Synchronous mode: before the new page is displayed, there is nothing the user can do on the existing page except wait!

Main Differences

- Ajax:
  - if only a small part of the current page requires update (e.g., the total price of the selected goods), a client request is made to the server. That request is handled *asynchronously* by the browser.
    - This means while the request is processed by the server and before the response arrives, the user can continue to work on the existing page (eg, filling the form).
  - Fast: because only small amount of data relevant to the update is retrieved from the server and only that small part of the current page is updated and re-rendered.
  - User interaction with the web application is not interrupted by the data retrieval operation (which can be slow), giving user a feeling of continuity and responsiveness.

Ajax Basics

- The most important object for writing an Ajax application is the `XMLHttpRequest` object.
- You create an `XMLHttpRequest` object (known as XHR object) using JavaScript to
  - create an HTTP request
  - define the callback function that will be called by the browser to process the response to the request
  - send the request to the server
  - the browser asynchronously
    - receive the response from the server
    - and then process the response (update the current page using the data retrieved from the server) by calling the registered callback function.

XMLHttpRequest

- A number of important properties and methods of `XMLHttpRequest` object are listed below. For full details, see W3C's working draft published on 30 January 2014.
- Properties:
  - `readyState`
  - `onreadystatechange`
  - `status`
  - `responseText`
- Methods
  - `open`
  - `send`

Creating XMLHttpRequest Object

- The first thing you need to do is create an `XMLHttpRequest` object:

```
var xhr = new XMLHttpRequest();
```

States of an XMLHttpRequest Object

- An `XMLHttpRequest` object goes through the following five states: UNSENT, OPENED, HEADERS_RECEIVED, LOADING and DONE.
- Each state has a symbolic name and a numeric value from 0 to 4. For instance, state DONE’s numeric value is 4.
- You can query the object for its current state using property `xhr.readyState`.

The Five States of XMLHttpRequest Obect

- `UNSENT` (`0`): the `XMLHttpRequest` object is constructed.
- `OPENED` (`1`): the open method is successfully called (hence the HTTP request is created). During this state, the `send` method can be invoked to send the HTTP request.
- `HEADERS_RECEIVED` (`2`): response headers are received from the server.
- `LOADING` (`3`): the response body is being received from the server.
- `DONE` (`4`): the complete response is received

Creating an HTTP Request

- To create an HTTP request using xhr object:
  - You need to state which HTTP method is used: GET or POST.
  - You need to specify the url of the server-side script that would process the request and generate the response.
  - Create the request with open method:

```
xhr.open( HTTP-method, server-script-url, asyn);
```

  - The third, optional, parameter `asyn` indicates whether to handle the request asynchronously (`true`) or not (`false`). Default is `true`.
- Example:

```
xhr.open( “GET”, “getCityState?zip=”+zip, true);
```

Sending the HTTP Request

- Send the request using

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

- The optional argument `data` is used to send data to the server. If you use HTTP GET method, the argument must be `null`, because the content part of the request message is empty.
  - Example:

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

- If you use HTTP POST method, you must provide data (such as user input from an HTML form). The data forms the content of the request message.

Registration of a Callback

- Before sending the request, you must register a callback function with the XHR object so that when the response comes back, the browser knows which function to call (automatically) to process the response.
- Define and register the callback using

```
 xhr.onreadystatechange = function () {
     // callback function body
 }
```

- Whenever `xhr.readyState`'s value changes, `onreadystatechange` event is triggerred, and the above callback will be called automatically.

Getting the HTTP Response

- Once the response is received by the web browser, it can be obtained via the following properties and methods of the XHR object:
  - `xhr.status`: status code of the response
  - `xhr.getResponseHead(header)`: return the header value(s) of the matching response header
  - `xhr.responseText`: the response body

Example 2: Loading File From Server

- Our first Ajax example is to load a file from the server. The response is in plain text.
- The example is available in directory `Examples/ex2`.

Example 2: Loading Plain Text File From Server

```
<script>
function loadFile(){
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function(){
        if (xhr.readyState == 4 && xhr.status == 200) {
            document.getElementById("demo").innerHTML
                    = xhr.responseText;
        }
    }
    xhr.open("GET", "ajax.txt", true);
    xhr.send();
}
</script>
```

Example 2: Loading File From Server

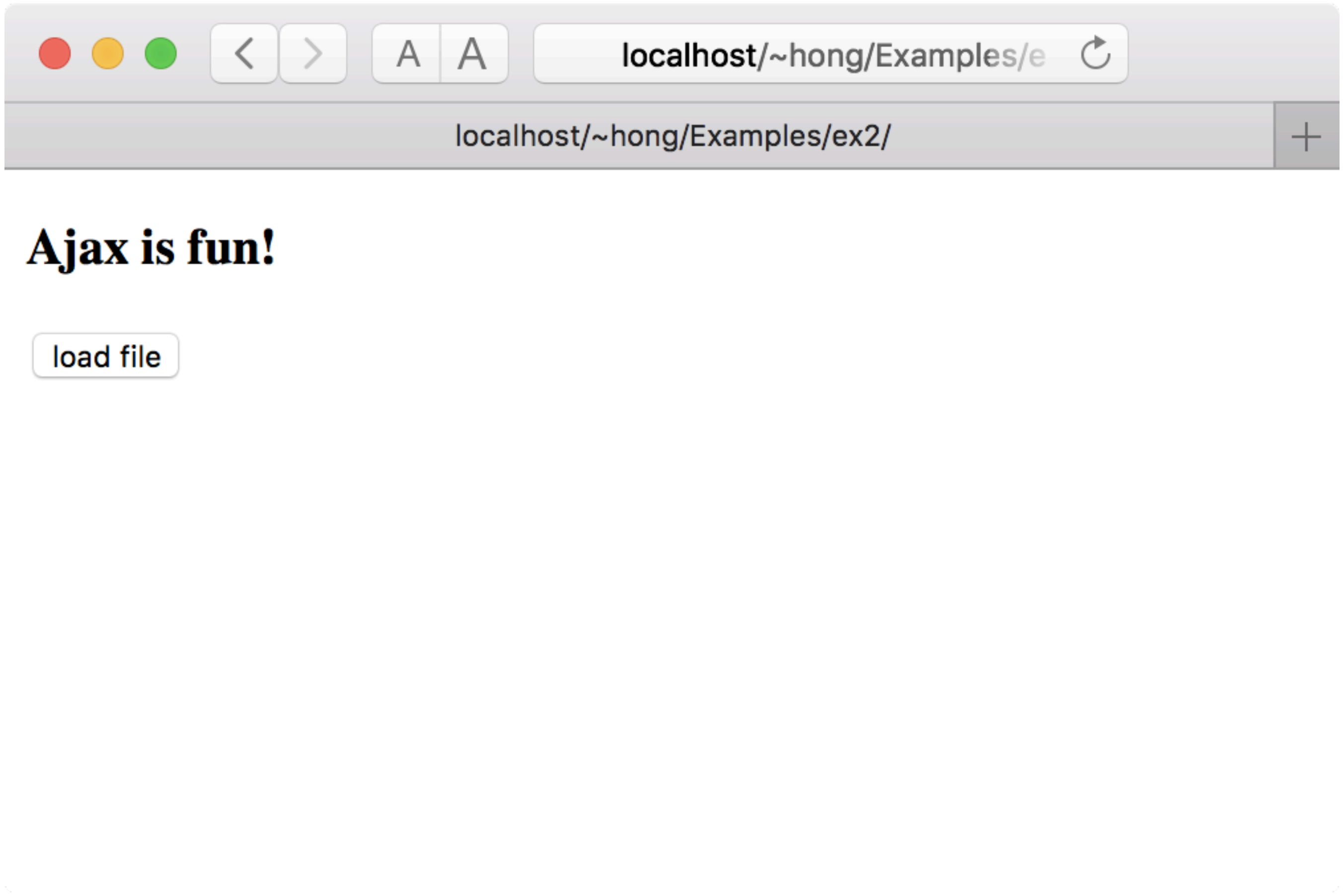
```
<body>
<div id="demo"></div>
<button onclick="loadFile();">load file </button>
</body>
```

The file ajax.txt:

```
<h3> Ajax is fun!</h3>
```

Note the response `this.responseText` is in plain text.

Example 2: Loading File From Server



Ajax is fun!

Example 3: Loading XML File From Server

- Our second Ajax example is to load an XML file from the server. The response is an XML object.
- The example is available in directory `Examples/ex3`.
- Note JavaScript supports XML DOM interface, as shown in this example.

Example 3: Loading XML File From Server

```
<script src="https://code.jquery.com/jquery-
3.3.1.min.js"></script>

<script>
   function loadXML(){
       var xhr = new XMLHttpRequest();
       xhr.onreadystatechange = function(){
           if (this.readyState == 4 && this.status == 200) {
               processFruits(this.responseXML);
           }
      }
      xhr.open("GET", "fruits.xml", true);
      xhr.send();
   }
```

Example 3: Loading XML File From Server

```
    function processFruits(xml) {
        var txt = '<table>';
        var fa = xml.getElementsByTagName('fruit');
        var i, name, price;
        for (i=0; i<fa.length; ++i) {
            name=fa[i].getElementsByTagName('name')[0].childNodes
[0].nodeValue;
            price=fa[i].getElementsByTagName('price')[0].childNod
es[0].nodeValue;
            txt += '<tr><td>'+ name + '</td><td>' + price +
'</td></tr>';
        }
        txt += '</table>';
        $("#demo").html(txt);
    }
</script>
```

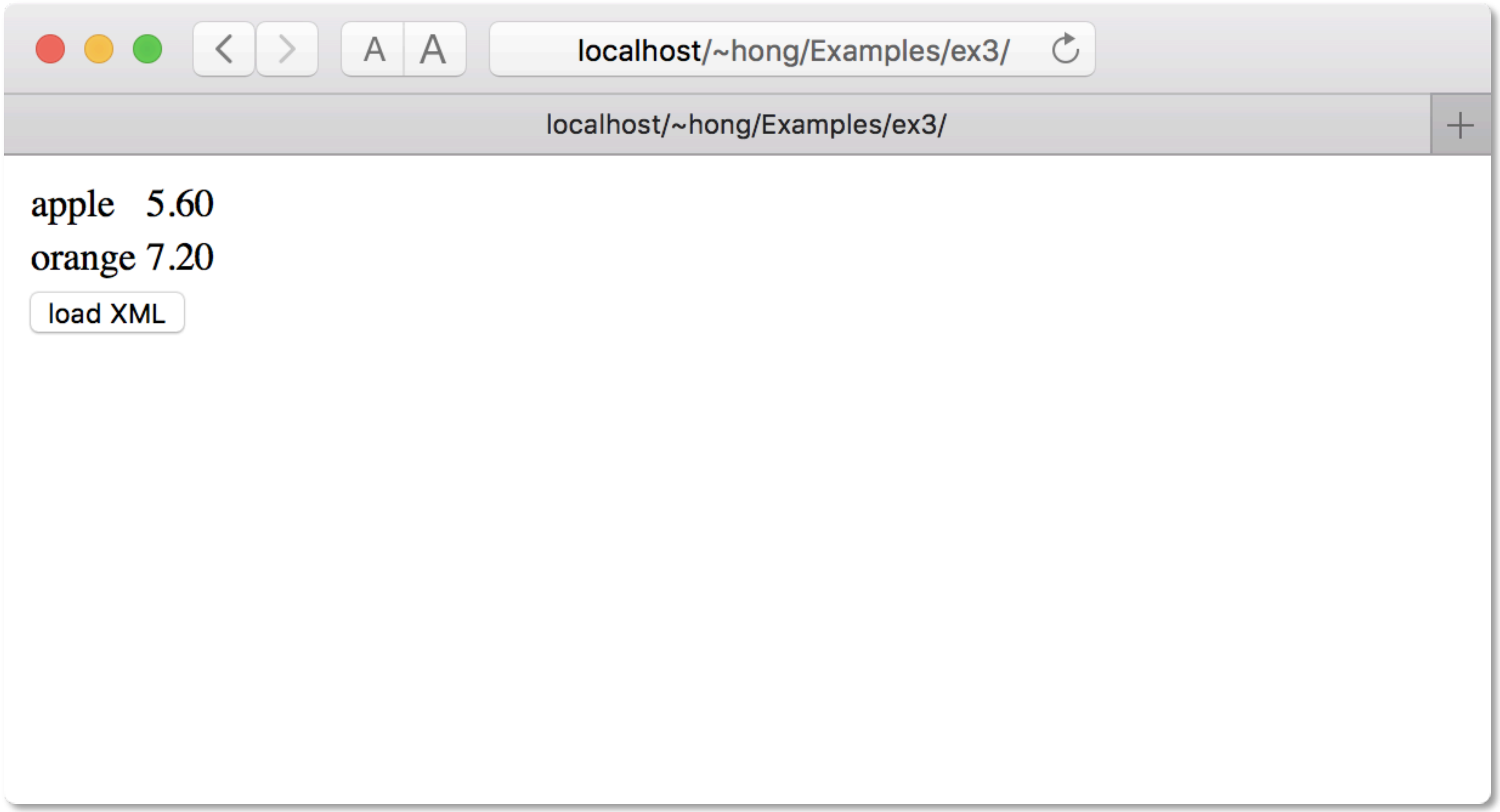
Example 3: Loading XML File From Server

```
<body>
<div id="demo"> </div>
<button onclick="loadXML();">load XML</button>
</body>
```

the file `fruits.xml` contains

```
<fruit-list>
<fruit> <name> apple </name> <price> 5.60 </price>
</fruit>
<fruit> <name> orange</name> <price> 7.20  </price>
</fruit>
</fruit-list>
```

Example 3: Loading XML File From Server



```
apple   5.60
orange 7.20
```

Example 4: Get A Reply From PHP Script

- Our third Ajax example is a client-server example. The client sends a name to the server, which is a PHP script.
- The server will check the name. If the name is same as my name, it replies "Hello Hong, how are you?". Otherwise it replies "Who are you?".
- This example is available in directory `Examples/ex4`.

Example 4: Get A Reply From PHP Script

```
<script>
function getGreeting() {
    var name=document.getElementById('name').value;
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function(){
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById('demo').innerHTML
                      = this.responseText;
        }
    }
    xhr.open("GET", 'greeting.php?name=' + name, true);
    xhr.send();
}
</script>
```

Example 4: Get A Reply From PHP Script

```
<body>
Name: <input type="text" id="name"/> <br/>
<div id="demo"></div> <br/>
<button onclick="getGreeting()">get greeting</button>
</body>
```

Example 4: Get A Reply From PHP Script

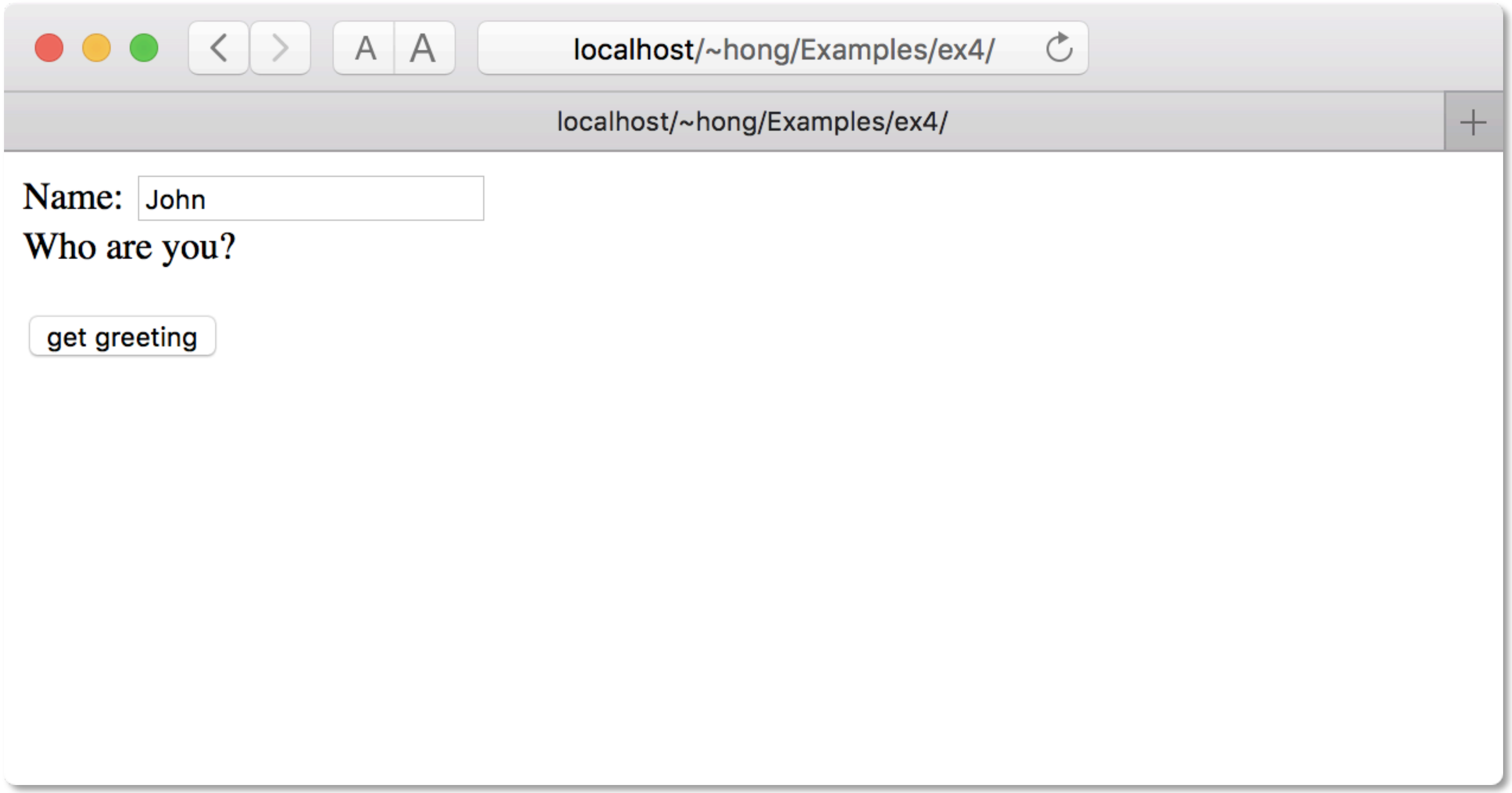
- The PHP script "greeting.php":

```
<?php

$name = $_GET['name'];
if ($name == 'Hong')
   echo "Hello Hong, how are you?";
else
   echo "Who are you?";

?>
```

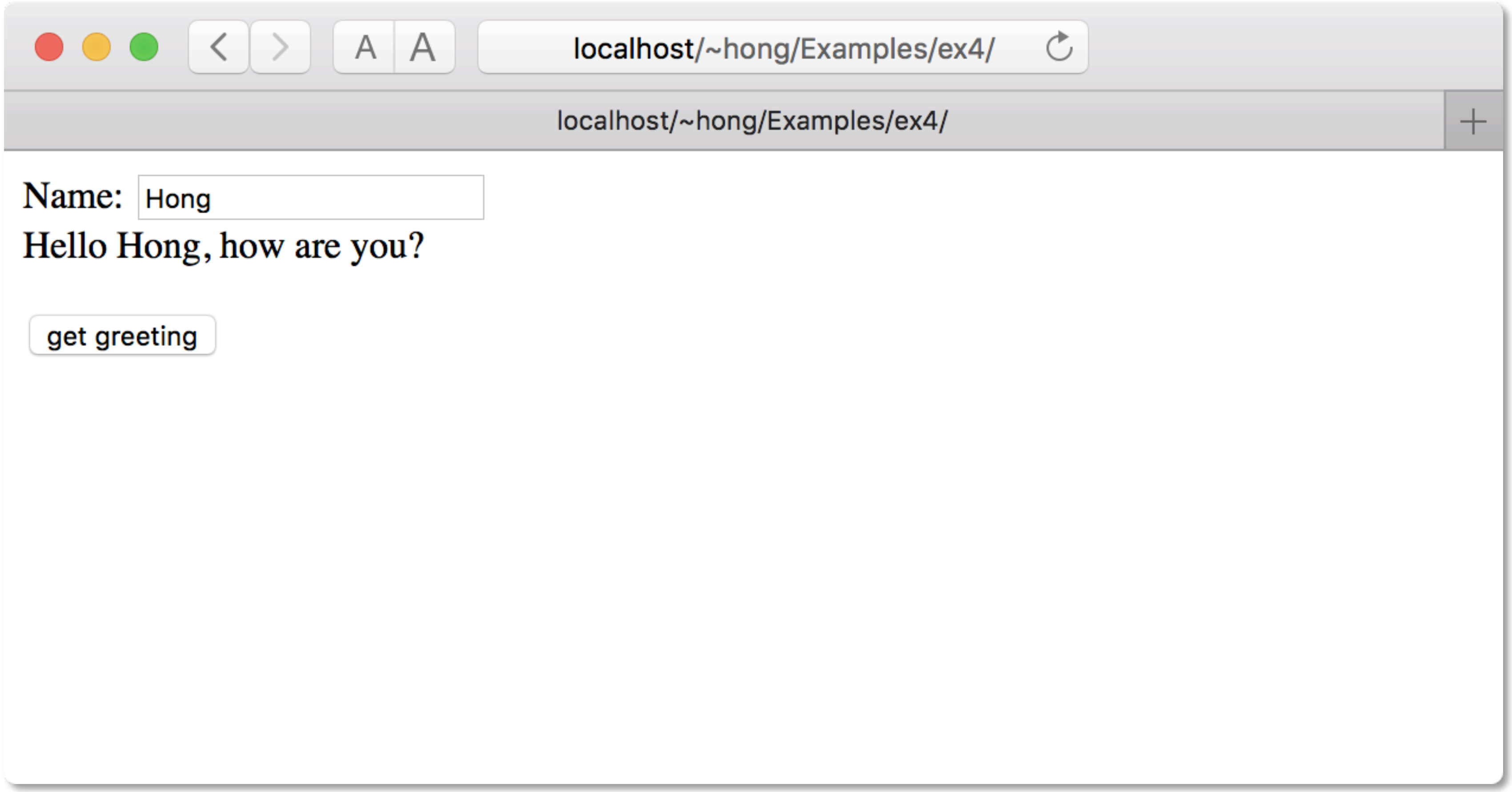
Example 4: Get A Reply From PHP Script



Name: John

Who are you?

Example 4: Get A Reply From PHP Script



Name: Hong

Hello Hong, how are you?

Response Formats

- With Ajax, the server response can be in one of several formats.
- For unstructured data, we can use plain text, including an HTML fragment.
- For structured data, we may use either XML or JSON.
- Most applications use JSON these days as it is simple and can be easily handled in JavaScript and PHP without extra software (unlike XML).

Replacing HTML

- We sometimes need to replace a fragment of the current HTML document with a piece of HTML code returned from the server.
- To do this, we may enclose the replaceable fragment within an element such as `<div>` element, as shown in Example 2.
- The DOM object for most element contains the `innerHTML` property.
- To replace the content of an HTML element with a new content, just assign the new content (a piece of text text) to `innerHTML`.

Replacing HTML

- Example:

Old list	New list

`<div id="units">`
`<p> List of Units </p>`
`<ul>`
`<li>MAS120</li>`
`<li>BUS230</li>`
`</ul>`
`</div>`
	`<div id="units">`
	`<p> List of IT Units </p>`
	`<ul>`
	`<li>ICT286</li>`
	`<li>ICT365</li>`
	`</ul>`
	`</div>`

- Assume the new list (in red) is returned from the server and is in `xhr.responseText`:

```
var divDom = document.getElementById("units");
divDom.innerHTML = xhr.responseText.
```

Handling XML

- The response message can be coded in XML to provide structured data.
- XML is an important technology for representing structured data.
- XML is a meta language that is used to create domain-specific markup language, such as MathML, ebXML, and XHTML.
- You can extract information from an XML message using either DOM binding parsing methods as shown in Example 3 or using XSLT.
- XML and the associated technologies are covered in ICT375 Advanced Web Programming.

JSON

- In HTTP, data exchanged between client and server must be in text format. They cannot be binary.
- However objects in JavaScript and PHP are binary data.
- One way to exchange objects between the client and server is to use JSON format.
- JSON stands for JavaScript Object Notation which is part of the JavaScript standard.
- Data encoded in JSON are text. Therefore they can be exchanged between HTTP client and server.

JSON

- JSON represents an objects as a text string, using two structures
  a) Collections of name/value pairs

     `{` *`name`*$_1$`:`*`value`*$_1$`,` *`name`*$_2$`:`*`value`*$_2$`, … }`

  b) Arrays of values

     `[` *`element`*$_1$`,` *`element`*$_2$`, … ]`
- In a *`name`*`:`*`value`* pair, the value itself can be an array or collection.
- Similarly, each array element itself can be another array or collection.
- You can have any number of the above nested structures in a JSON string.

JSON

- Example: the following notation represents the a collection that consists of one property/value pair, whose value is an array of three collections, each with two property/value pairs:

```
{"employees" :
  [
    {"name" : "Dew, Dawn", "address" : "1222 Wet Lane"},
    {"name" : "Do, Dick",  "address" : "332 Doer Road"},
    {"name" : "Deau, Donna", "address" : "222 Donne Street"}
  ]
}
```

JSON

- A JSON string is returned as a text string in `responseText`. It needs to be converted into JavaScript object in order for us to access its properties.
- The object could be obtained by running `eval` on the response string, but this is dangerous, because the response string could contain malicious code.
- It is safer to use JSON parser method `JSON.parse()` to convert a JSON string to a JavaScript object:

```
var response = xhr.responseText;
var myObj = JSON.parse(response);
```

- You can access data via myObj, eg,

```
var address2 = myObj.employees[1].address;
```

The above `address2` would contain “`332 Doer Road`”.

JSON

- Sometimes we have a JavaScript object that needs to be sent to the server.
- An object must be converted to text format before it can be sent to the server.
- This can be done with `JSON.stringify` method.
- Example:

```
var myObj = { name: "John",
          age: 25,  city: "Perth"};
var myJSON = JSON.stringify(myObj);
```

- We will provide more examples in Topic 10: Arrays, Ajax, jQuery, JSON and Cookies: More Examples.

Ajax References

- W3C XMLHttpRequest - W3C Working Draft:

  http://www.w3.org/TR/XMLHttpRequest/

- W3Schools Ajax Tutorial:

  http://www.w3schools.com/ajax/default.asp