COMP721 Web Development

AUT UNIVERSITY

Week 9: XML/JSON and Ajax/Fetch interaction cycle

Week 8 review
Server-Side Technology Overview
Introduction to XML and JSON
Client-Server Ajax Interaction Cycle
The shopping cart example

Review of Lecture 8 (JavaScriptDOM)

- Javascript
  - ☐ Dynamic typing, object-based, no access modifiers
  - ☐ Runs in a browser: you need to know the browser objects:

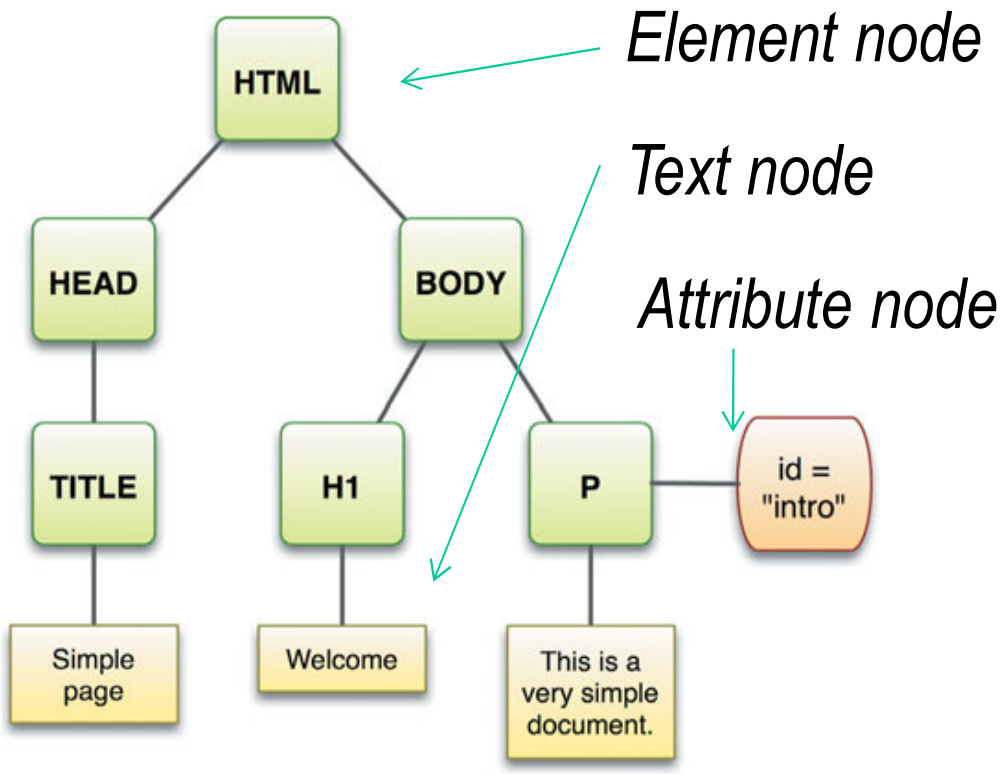
    Windows, history, location, navigator, screen, document…
  - ☐ Four methods to define user objects:

    Constructor function, Object constructor, Object literal, and Prototypes

```
function Member (name, emaddr, reg)
{ this.name = name;
  this.email = emaddr;
  this.isRegistered = reg;
}
Member.prototype.present = function () {
  alert("I'm here! ");
}; // the method is defined and shared for all member objects
…
m5 = new Member();
```

Review of Lecture 8 (JavaScriptDOM)

- DOM
  - DOM is an API for managing (CRUD) XML and XHTML documents
  - Three types of most frequent nodes in an XML document



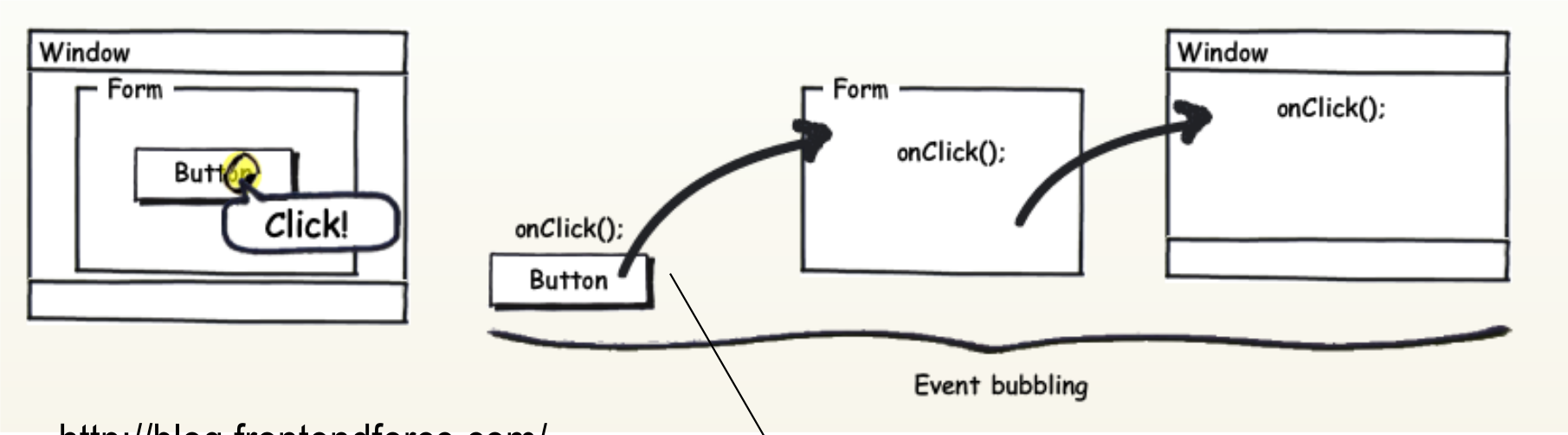
DOM Examples

- document.getElementsByTagName(“myElement”)**[0].**parentNode
- document.getElementById(“myId”).parentNode
- document.getElementById(“myId”).childNodes[i]
  document.getElementById(“myId”).childNodes.item(i)
- document.getElementById(“myId”).firstChild
- document.getElementById(“myId”).lastChild
- document.getElementsByTagName(“myElement”)[0]

  document.getElementsByTagName(“myElement”).item(0)
- document.getElementsByTagName("myElement").item(
  document.getElementsByTagName("myElement").childNodes.length - 1)

Review of Lecture 8 (JavaScriptDOM)

- JS event model
  - Bubbling and capturing



http://blog.frontendforce.com/

invoke

Callback function

Event Registration

- **Inline event registration** by using HTML attributes
  <a href="goThere.html" onclick="startNow(); return false">
- **Traditional event registration**
  var myElement = document.getElementById('1stpara');
  myElement.onclick = startNow;
  to remove myElement.onclick = null;
- **W3C DOM event registration**
  var myElement = document.getElementById('1stpara');
  myElement.addEventListener('onclick', startNow, false);
  myElement.addEventListener('onclick', startNow2, false); // additional
  to remove myElement.removeEventListener('onclick', startNow, false);

Agenda

- Server-Side Technology Overview
- A Brief Introduction to XML and JSON
- Client-Server Ajax Interaction Cycle
  1. Submission model: Traditional Form-based Model v.s. Ajax Model
  2. The server receives the data and generates XML response
  3. Receiving Data from the Server
- The Fetch Interaction Cycle
- The shopping cart example

Server-Side Technologies

- PHP
  - Popular, easier to learn
  - mostly used for small to medium applications
- Node.js
  - A Javascript/TypeScript runtime environment, gaining popularity
  - Fast and scalable
  - Supports full-stack web development with JS
- Java EE, ASP.NET
  - Good framework support, better for enterprise applications
  - mostly used for large scale applications
- Others: Perl, Python, Ruby…

XML

- In simple examples, TEXT format may be used to transmit data from the server to the client
- In serious examples, the data transmitted is usually structured
  - Both XML and JSON are structured data representation language
  - The knowledge about XML is applicable to JSON
  - HTML is a type of XML-based language
- XML is the “standard” structured data description language on the web
- Note that we can use our own method to process simply structured text – eg comma-delimited strings, which can be separated and processed on the client – but the features of XML offer far greater power and flexibility.

An XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Persons>
    <Person>
        <Name>
            <First>Thomas</First>
            <Last>Atkins</Last>
        </Name>
        <Age>30</Age>
    </Person>
    <Person>
        <Name>
            <First>Sachin</First>
            <Last>Tendulkar</Last>
        </Name>
        <Age>38</Age>
    </Person>
</Persons>
```

XML declaration

Json version

```
x = {


    "Persons" : [
            { "Name": {First: 'Thomas', Last:'Atkins'}, Age: 30},
            { "Name": {First: 'Sachin', Last:'Tendulkar'}, Age: 38}
    ]
};
```

Why use XML?

- Simplicity
- Extensibility
- Interoperability
- Openness
- Platform-independent, industry accepted standard
- One data, different views
- Internationalization (Unicode
- Extensive software tools available
- Industry-specific schemas
  - E.g., ebxml

JSON is more lightweight

XML is more versatile (go beyond the scope of web development)

XML vs HTML

- For now, think of an XML document as being similar in form to an HTML document, but with user-defined tags
- In HTML, tags are used to indicate formatting. In XML they are used to describe data.
- As with HTML, there is an XML DOM, which can be accessed using the same DOM model.
- XML can be processed on the client and the server.

XML vs. HTML : Simple Example

```
<TABLE>
  <TR>
    <TD>Thomas</TD><TD>Atkins</TD>
  </TR>
  <TR>
    <TD>age:</TD><TD>30</TD>
  </TR>
</TABLE>
```

HTML, using pre-defined tags which have pre-defined meanings wrt presentation in a browser

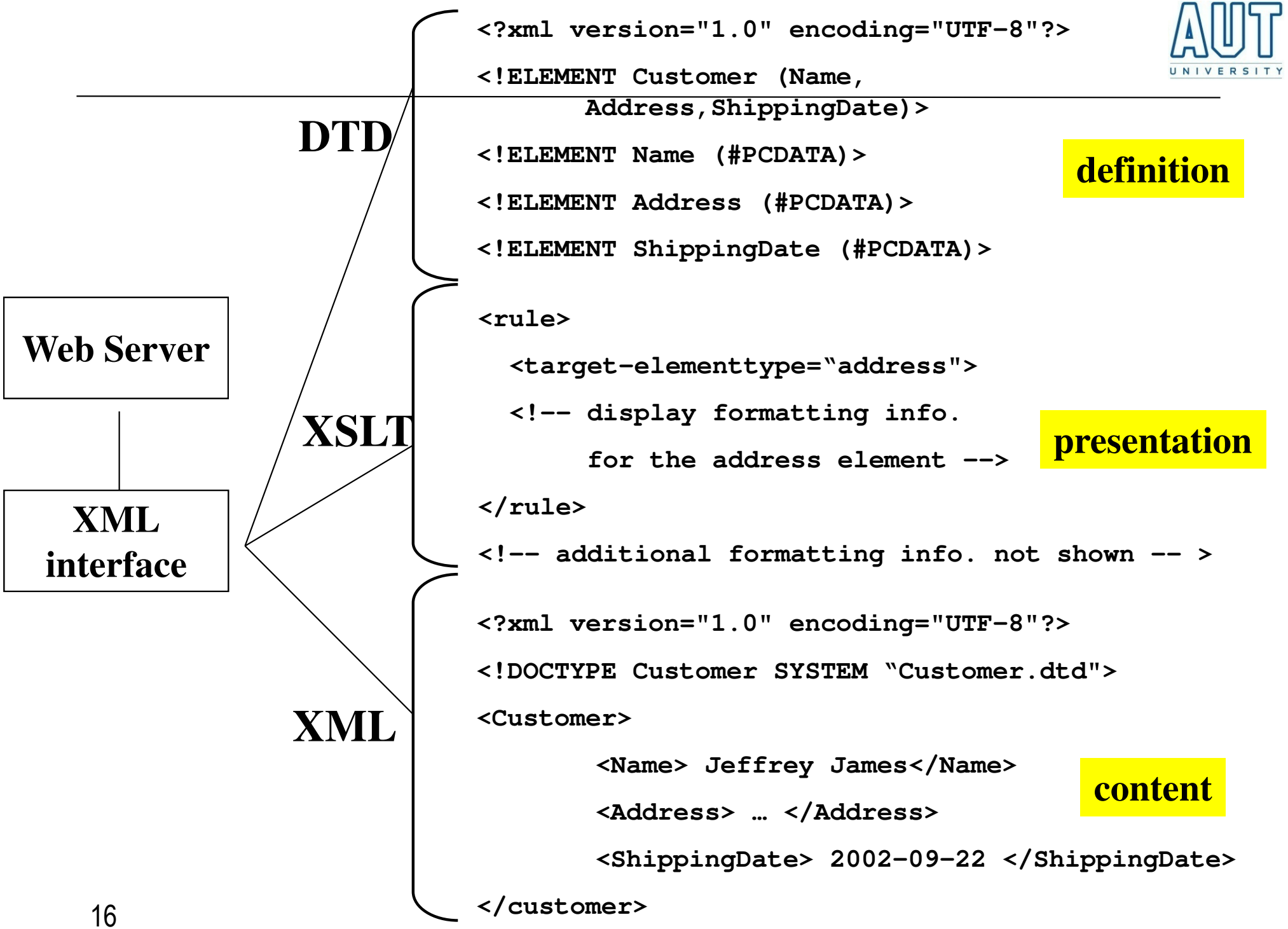
```
<Person>
  <Name>
    <First>Thomas</First>
    <Last>Atkins</Last>
  </Name>
  <Age>30</Age>
</Person>
```

XML, with user-defined tags which are chosen to convey intent of their content

Three major XML Components

For an XML document to be easy to be interpreted
it must contain three distinct parts:

- the content: that makes up the information contained within the document.
- the schema (similar to DB table schema): definition of the valid element names and structure of document elements using XML DTDs or Schema;
- the presentation of a document's visual aspects, e.g., its style defined by means of Stylesheets.


Web Server
XML interface
DTD
XSLT
XML
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Customer (Name,
Address,ShippingDate)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT ShippingDate (#PCDATA)>
definition
<rule>
<target-elementtype="address">
<!-- display formatting info.
for the address element -->
</rule>
<!-- additional formatting info. not shown -- >
presentation
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Customer SYSTEM "Customer.dtd">
<Customer>
<Name> Jeffrey James</Name>
<Address> … </Address>
<ShippingDate> 2002-09-22 </ShippingDate>
</customer>
content

Well-formed and Valid XML

- XML is well-formed if it conforms to certain general rules (eg, all elements must be closed off; elements must nest; there must be a single ‘root’ element).
- XML is valid with respect to a DTD if it satisfies the requirements of the DTD.

XML Syntax rules – wellformedness

- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- XML Documents Must Have a Root Element
- XML Attribute Values Must be Quoted
- Entity References

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

There are 5 predefined entity references in XML: See the table at left top

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

XML DTD – validness

- For **validating** xml documents
  - What elements must be included
  - The order of the elements
  - The number of times each element can occur in the document
  - Any attributes of the elements

Note: XML Schema can also define what type of data can be contained in an element

A DTD for the XML

Persons.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Persons ((Person+))>
<!ELEMENT Person ((Name, Age))>
<!ELEMENT Name ((First, Last))>
<!ELEMENT Last (#PCDATA)>
<!ELEMENT First (#PCDATA)>
<!ELEMENT Age (#PCDATA)>
```

PCDATA:
Parsed char Data
e.g., '<' will be translated to '<' for rendering

DTD – elements

- <!ELEMENT br EMPTY>:
- <!ELEMENT from (#PCDATA)>
- <!ELEMENT note ANY>
- <!ELEMENT note (to,from,heading,body)> : in sequence
- <!ELEMENT note (message)>: one occurrence
- <!ELEMENT note (message+)>: more than one
- <!ELEMENT note (message*)>: zero or more
- <!ELEMENT note (message?)>: zero or one
- <!ELEMENT note (#PCDATA|to|from|header)*>: mixed content

XML for Persons with a DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Persons SYSTEM “Persons.dtd">
<Persons>
    <Person>
        <Name>
            <First>Thomas</First>
            <Last>Atkins</Last>
        </Name>
        <Age>30</Age>
    </Person>
    <Person>
        <Name>
            <First>Sachin</First>
            <Last>Tendulkar</Last>
        </Name>
        <Age>38</Age>
    </Person>
</Persons>
```

Using XML in Ajax

- structured data can be:
  - assembled or stored in XML form on the server, use being made of the DOM API or other APIs
  - processed using a server-side language on the server (eg, PHP DOM API, XSLT)
  - sent from server to client (to the responseXML property of an XHR object)
  - processed using JavaScript on the client (using the DOM API)

Using JSON in Ajax

- …
- sent from server to client (to the responseText property of an XHR object)
- System automatically convet JSON string to an object:

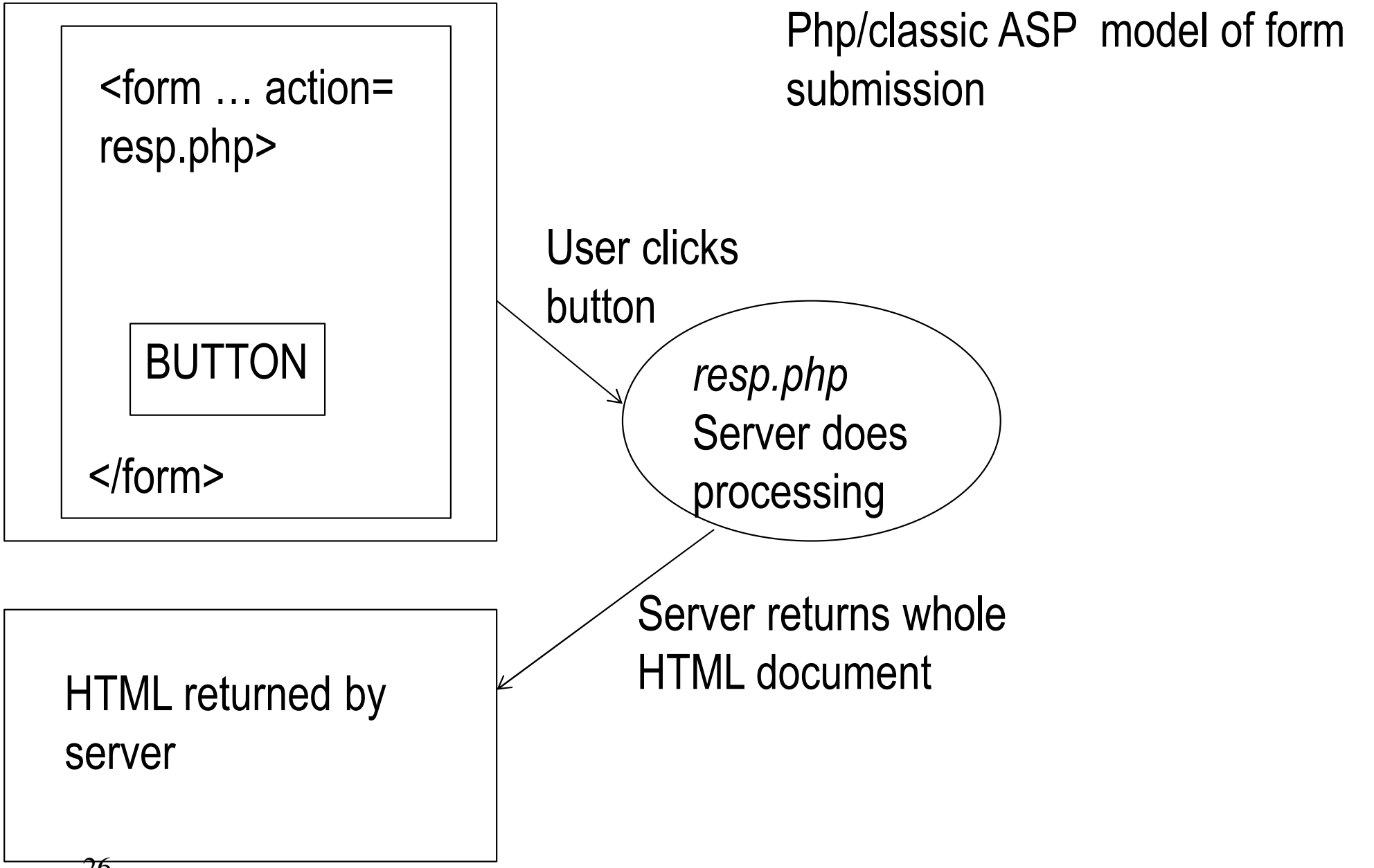
```
var myObj = JSON.parse(this.responseText);
```

The Ajax interaction cycle

1. Submission model:

Traditional Form-based Model v.s. Ajax Model

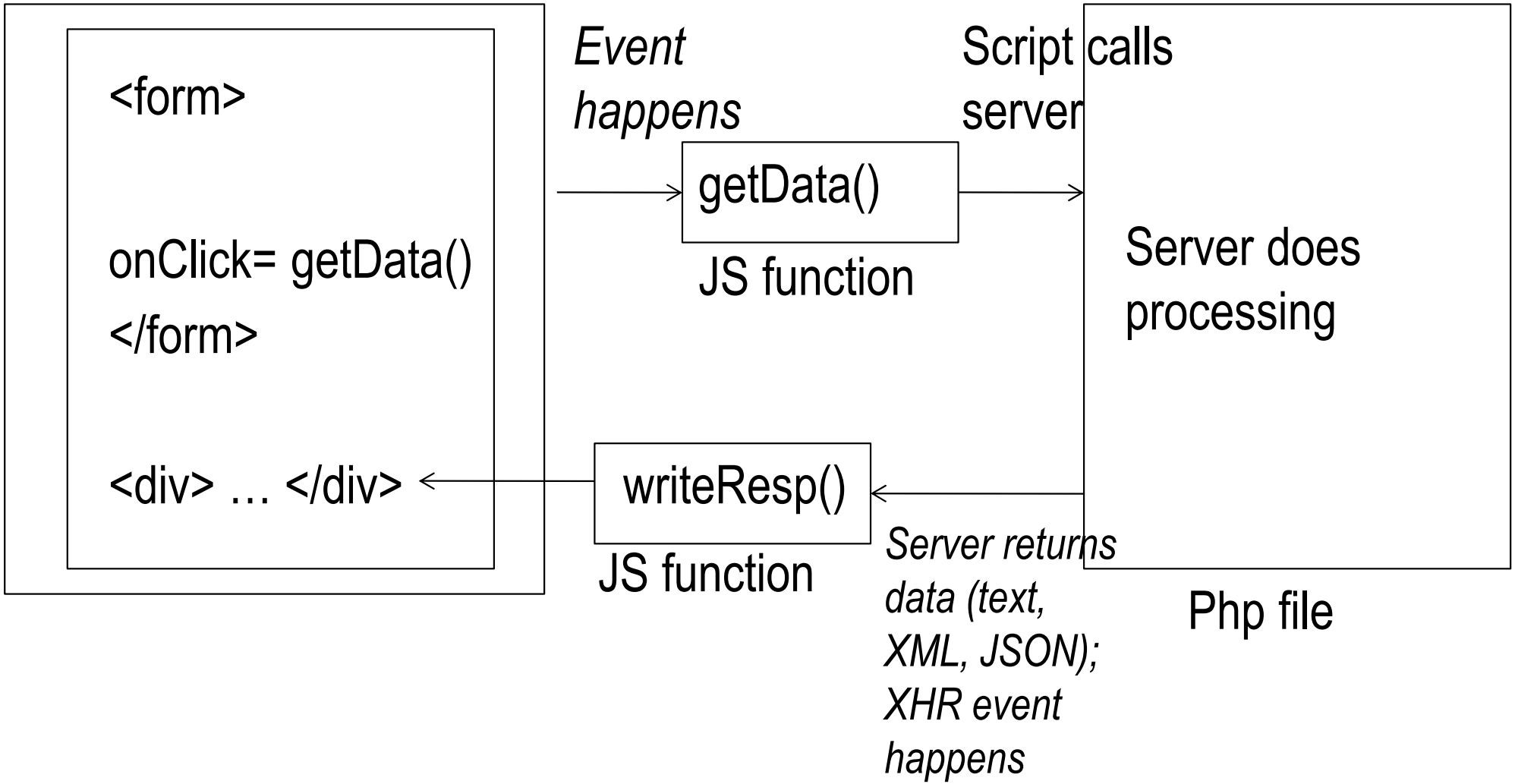
Traditional Form-based Model



Php/classic ASP model of form submission

Ajax/Fetch model of submission

NB: the model is event-driven, so the form can be completely removed from the model



Submitting Data to the Server – Ajax

- Three steps, as seen in example in Lecture 7:
  1. Call the *open* method on the XHR object with the request to establish the client-server link
  2. Specify the call-back function that will execute on the client when the client gets notification that the state of the XHR object has changed
     - *Guarded by a conditional that ensures processing only occurs when the state has been changed to "loaded"*
  3. Send the request, to activate the communication
- Note that the client keeps being open for further user-interaction whilst the server processes the request (which may take some time to complete)

Submitting Data to the Server – Ajax

- Assume **xhr** has been defined as an XMLHttpRequest object
- *open* method

```
xhr.open(method, URL to call, asyn or syn)
```

"true" means asynchronous; "false" means synchronous

- Two request methods
  - HTTP *Get*

```
xhr.open("GET", "response.php?value=1", true);
xhr.send(null);
```

  - HTTP *Post*

```
var argument = "value="; argument += encodeURIComponent(data);
xhr.open("POST", "response.php", true);
xhr.send(argument);
```

The Ajax interaction cycle

2. The server receives the data and generates XML response

The Server Receives the Request

- The request sent by **GET** or **POST** is part of either the **URL** (in the case of GET) or the ***Request* body** (in the case of POST)
- The server picks up these items as variables
  - PHP uses $_*GET*, $_*POST*

Client

```
<input type="text" name="MyTextbox1" />
```

Server

```
$Tb = $_GET["MyTextbox1"];

$Tb = $_POST["MyTextbox1"];
```

Constructing and Writing the HTTP Response in XML

File XML.php

```
<?
$doc = new DomDocument('1.0');
$root = $doc->createElement('ajax');
$doc->appendChild($root);
$child = $doc->createElement('js');
$root->appendChild($child);
$value = $doc->createTextNode("coordination");
$child->appendChild($value );
$strXml = $doc->saveXML();
echo $strXml;
?>
```

XML created

```
<?xml version="1.0"?>
<ajax>
   <js>coordination</js>
</ajax>
```

- $doc is a PHP variable that represents a structured XML document
- The saveXML() method returns the string representation of that object
- echo expects a string as its argument

Constructing and Writing the HTTP Response in JSON

- Inoke the “json_encode() function
- The object to be serialized could be of any complexity, e.g., nested arrays, objects, primitives…

```
echo json_encode($anObj,
JSON_PRETTY_PRINT);
```

```php
<?php
$myObj->name = "John";
$myObj->age = 30;
$myObj->city = "New York";

$myJSON = json_encode($myObj);

echo $myJSON;
?>
```

output

```
{"name":"John",
"age":30,
"city":"New York"}
```

Notes on the Some of the Code

```
<?
1:$doc = new DomDocument('1.0');
2:$root = $doc->createElement('ajax');
3:$doc->appendChild($root);
```

- 1: creates a new Dom document and assigns it to variable $doc
- 2: creates a new element and assigns it to the variable $root
  - Note that createElement is a method of the DomDocument class in PHP, and it creates a new element for the DomDocument object.
- 3: appends $root as a child of the $doc element. (Note that this method call does return a value – namely the node, $root, that was appended to $doc, but we just ignore the return value and treat the method as if it had been declared as a void method.)

Send XML documents to the client

- Send XML documents to the client
  - Just "echo" the text form of an XML document, which loads the document to the XHR object on the client
- We illustrate the manipulation of XML using PHP on the server by a simple example (next slide)
- If this PHP file was "called" in an Ajax application, we can access the returned XML document using responseXML property of the XHR object

XML on the Server

- New XML documents (see example)
  - Create an XML DOM object (construct method)
  - Then add various nodes by XML DOM API
- Existing XML documents
  - Load an XML document stored on the server (load method)
  - Read or update nodes by XML DOM API
- Store XML documents
  - Save an XML document on the server (save method)
  - “Serialise” an XML document as a string (saveXML method)

JSON on the Server

- JSON is just plain text
- Load JSON from a file:
  - `$jsondata = file_get_contents($myFile);`

    `$obj = json_decode($jsondata, true);`
- Save JSON to a file:
  - `$jsondata = json_encode($arr_data, JSON_PRETTY_PRINT)`

    `file_put_contents($myFile, $jsondata)`

The Ajax interaction cycle

3. Receiving Data from the Server

Receiving Data from the Server

- The data sent from the server will be received by the XHR object.
- To access this for processing on the client, we have to set up a callback function in the client-side JavaScript program.
- The function should check *readyState* of the XHR object to see if the data load is complete (*readyState* = 4)? If yes, we also check the status of the XHR object to see if the data transmission has been completed without error (eg, status = 200).
- If readyState is not 4, do nothing and wait for the next state change that calls the function again. Else we check the status, and if it is 200, then we proceed to process the received data on the client.
- There are two properties of the XHR object to choose to examine data from server
  - responseText – holds the response as a string
  - responseXML – holds the response as an XML DOM object (which can be manipulated as such in JavaScript)

Receiving Data from the Server

- Assume that method getData has been set as the call-back function on the readyStateChange event on the XHR object; thus it is called every time the readyState property changes its value.
- **We only want the processing to occur when the state has changed to value 4.** The function will be called many times before the desired final state is reached (the state will change from 0 to 1 to 2 to 3 and then to 4). We include an "if" statement that only processes the data when we have assurance that the data is there! (In the *simpleajax* example, we had the call-back function alert the value of xhr.readyState, so that we could see the progression of its value from 0 through to 4 with successive event occurrences.)

Standard pattern in call-back function

```
function placeData()
{
  if ((xhr.readyState == 4) && (xhr.status == 200))
  {
    // processing here
  }
}
```

XHR ‘redayState’ property

Value	State	Description
0	UNSENT	Client has been created. open() not called yet.
1	OPENED	open() has been called.
2	HEADERS_RECEIVED	send() has been called, and headers are available.
3	LOADING	Downloading; responseTex t holds partial data.
4	DONE	The operation is complete.

Receiving Data from the Server

- Alternatively we can nest the tests in the condition in the call-back function, so that if the status is not 200, we can signal that this is the issue (generally it means that something has gone wrong):

```
function placeData()
{
  if (xhr.readyState == 4)
      if (xhr.status == 200)
      {
        // data received and ok : process it
        // put the code for processing the data here
      }
     else
     {
         alert (“status problem”);
      }
}
```

The responseText Property

- At the client side, use a JavaScript variable to collect the contents of the response from xhr.responseText if it has been sent as a text string

```
// client side
var text = xhr.responseText;
```

- At the server side, use PHP to prepare the data, and send as a text string

```
// PHP at the server side
$data = “This is returned data”;
echo $data;
```

- We can use responseText to retrieve a serialised XML document, but will lose many benefits – it is not in the form of a DOM document, ready to be processed by JavaScript.

The responseXML Property

- At the server side, use either PHP (or another server-side language) to prepare the XML document

```
// PHP at the server side
$data = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"
standalone=\"yes\"?> <root><child>Data</child></root>";
header("Content-type: text/xml");   // have to set this for IE
echo $data;
```

- At the client side, use a JavaScript variable to collect the contents of the *response from xhr.responseXML*

```
// client side
var myDoc = xhr.responseXML;
```

```
// client side using Firefox/Chrome
xhr.overrideMimeType("text/xml"); // if ContentType is not set at server side
xhr.send(null);                   // assuming GET protocol
……..
var myDoc = xhr.responseXML;
```

Fetching Data from responseXML

- Use DOM API in JavaScript to access nodes

```
<?xml version="1.0"?>
<cart>
  <book>
    <Title>Beginning Ajax</Title>
    <Qty>1</Qty>
  </book>
</cart>
```

```
var myCart = xhr.responseXML;
var books =
myCart.getElementsByTagName("book");
var titleNode = books[0].firstChild;
var qtyNode = titleNode.nextSibling;
```

```
// now extract the actual title – in IE (4~10)
var title = titleNode.text
```

```
// now extract the actual title in WC3 compliant
// browser
var title = titleNode.textContent
```

Debugging responseXML

If responseXML gets no content :

- Step 1: Check if there is content using responseText

```
var text = xhr.responseText;
alert(text);
```

- Step 2
  - IE: use errorCode for the error code and reason for error message

```
var errorCode = xHRObject.responseXML.parseError.errorCode;
var errorMessage = xHRObject.responseXML.parseError.reason;
```

  - Chrome/Firefox: use JavaScript Console, or Firebug to detect what may be wrong

The Fetch interface

```
function getData(dataSource, divID, aName, aPwd)  {

var place = document.getElementById(divID);
var url = dataSource+"?name="+aName+"&pwd="+aPwd;
const requestPromise = fetch(url);
  requestPromise.then(
    function (response){
    response.text().then(function(text) {
      place.innerHTML = text;
    });
  });
}
```

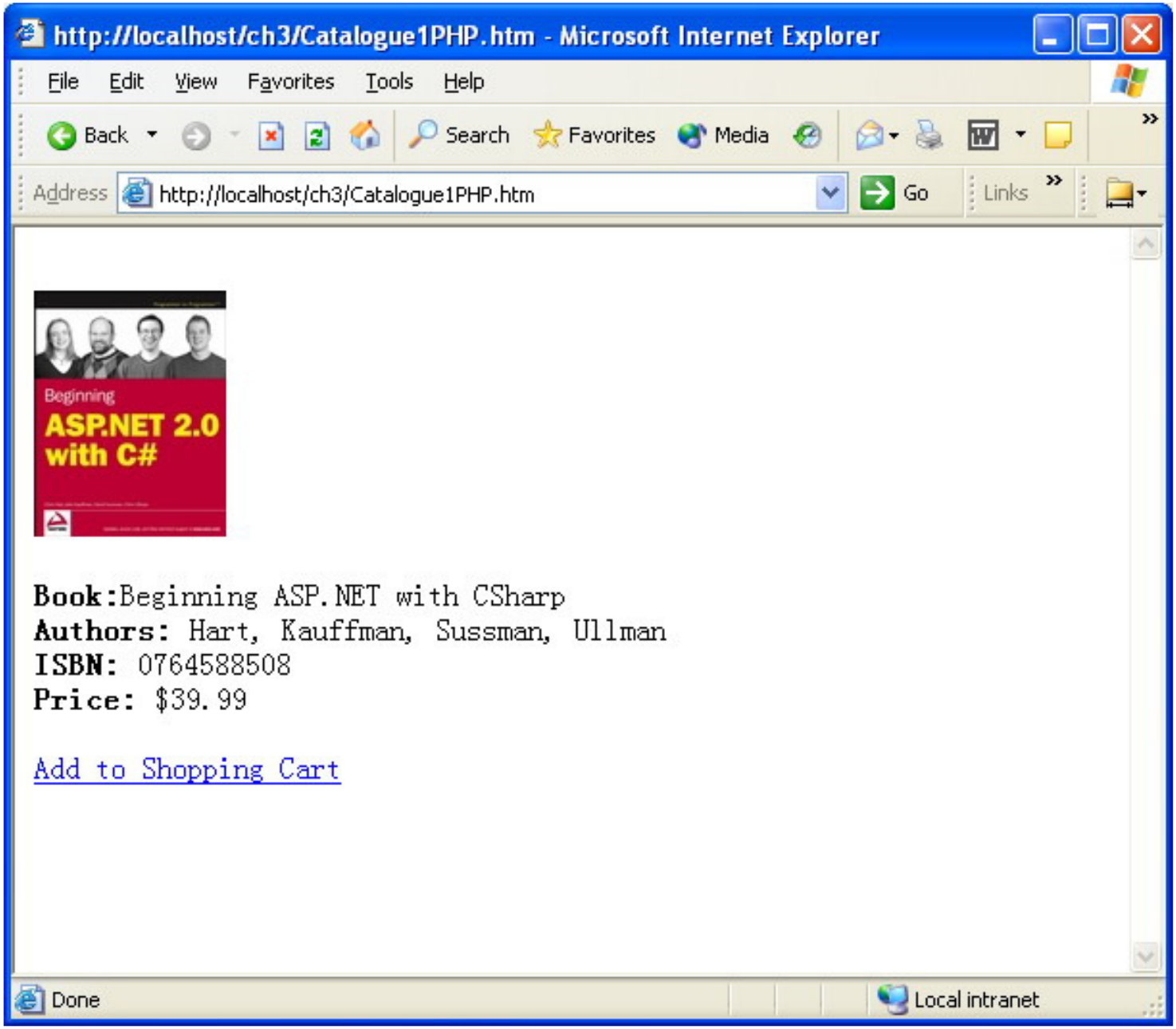
The shopping cart example

Both XML and JSON versions are provided on Blackboard

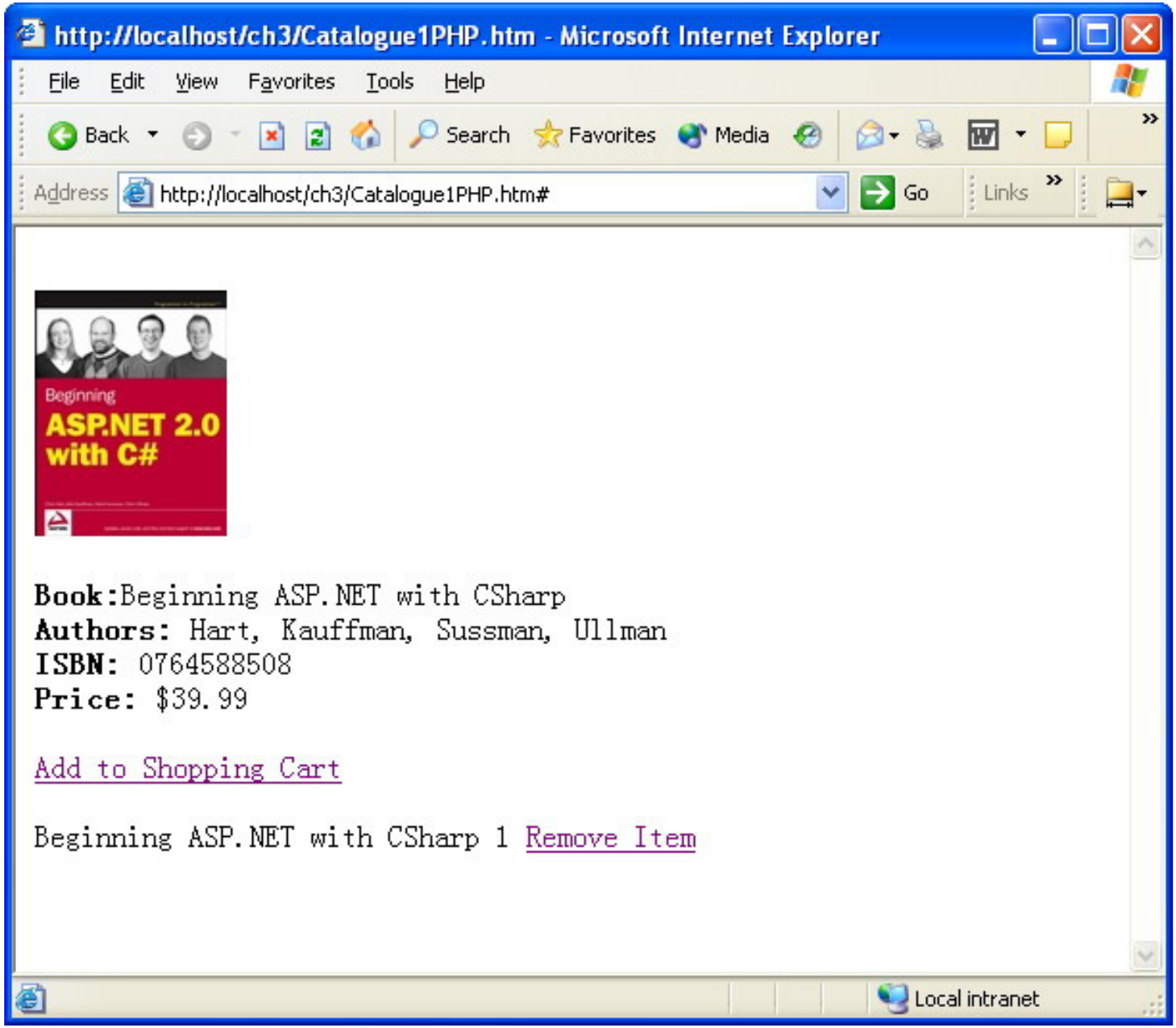
Shopping Cart Example

- Create a dummy catalogue page for a book seller using a shopping cart (the book seller sells just ONE book, for now)
- Allow users to place items in the shopping cart
- Allow users to update the shopping cart without the need to refresh the page (buy additional; remove all)
- Assume the user has been identified
- The shopping cart has three simple functions
  - Can add new items to it
  - The quantity will increase by one if a second item is added
  - Can remove all items from it

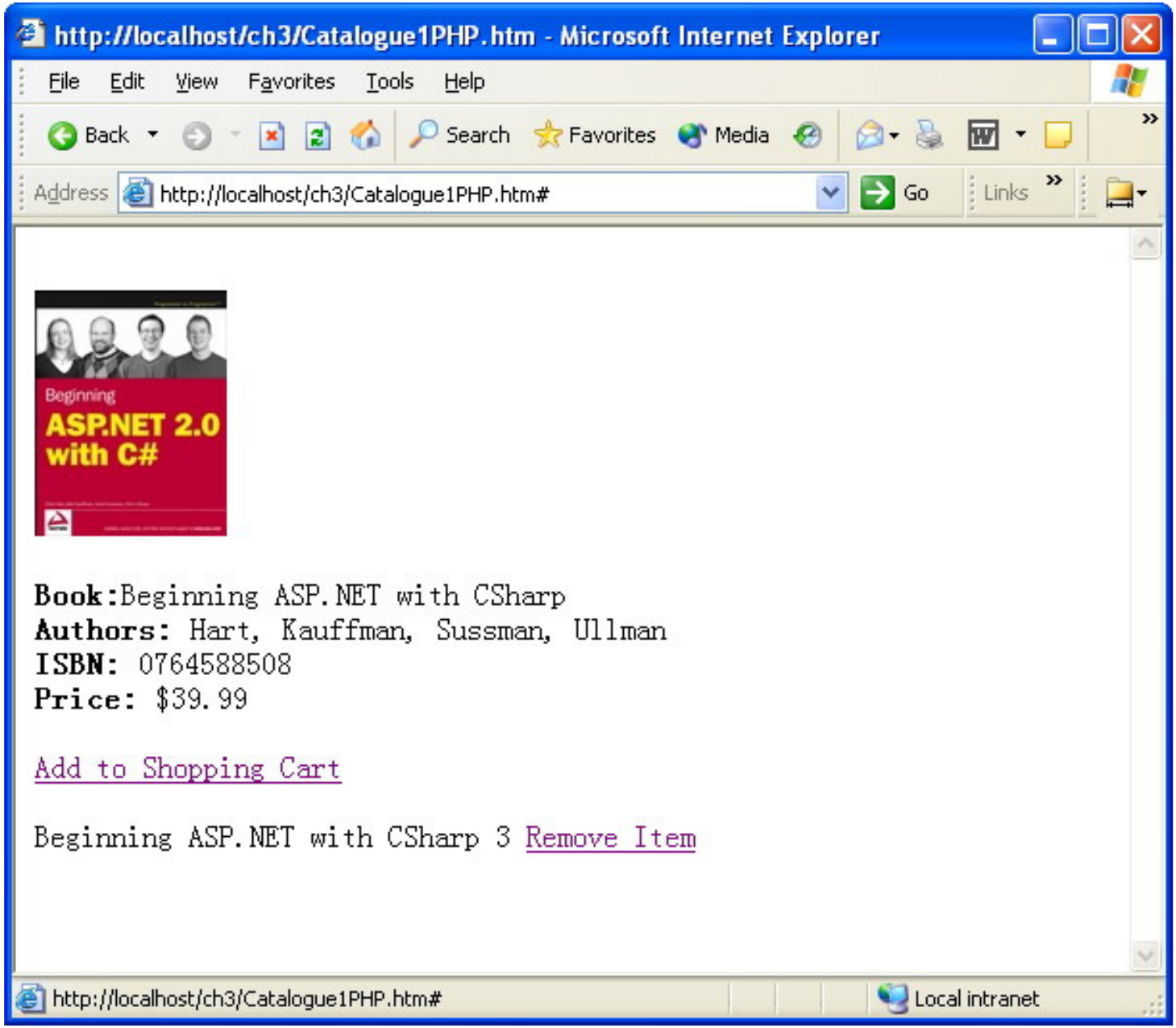
Shopping Cart Example



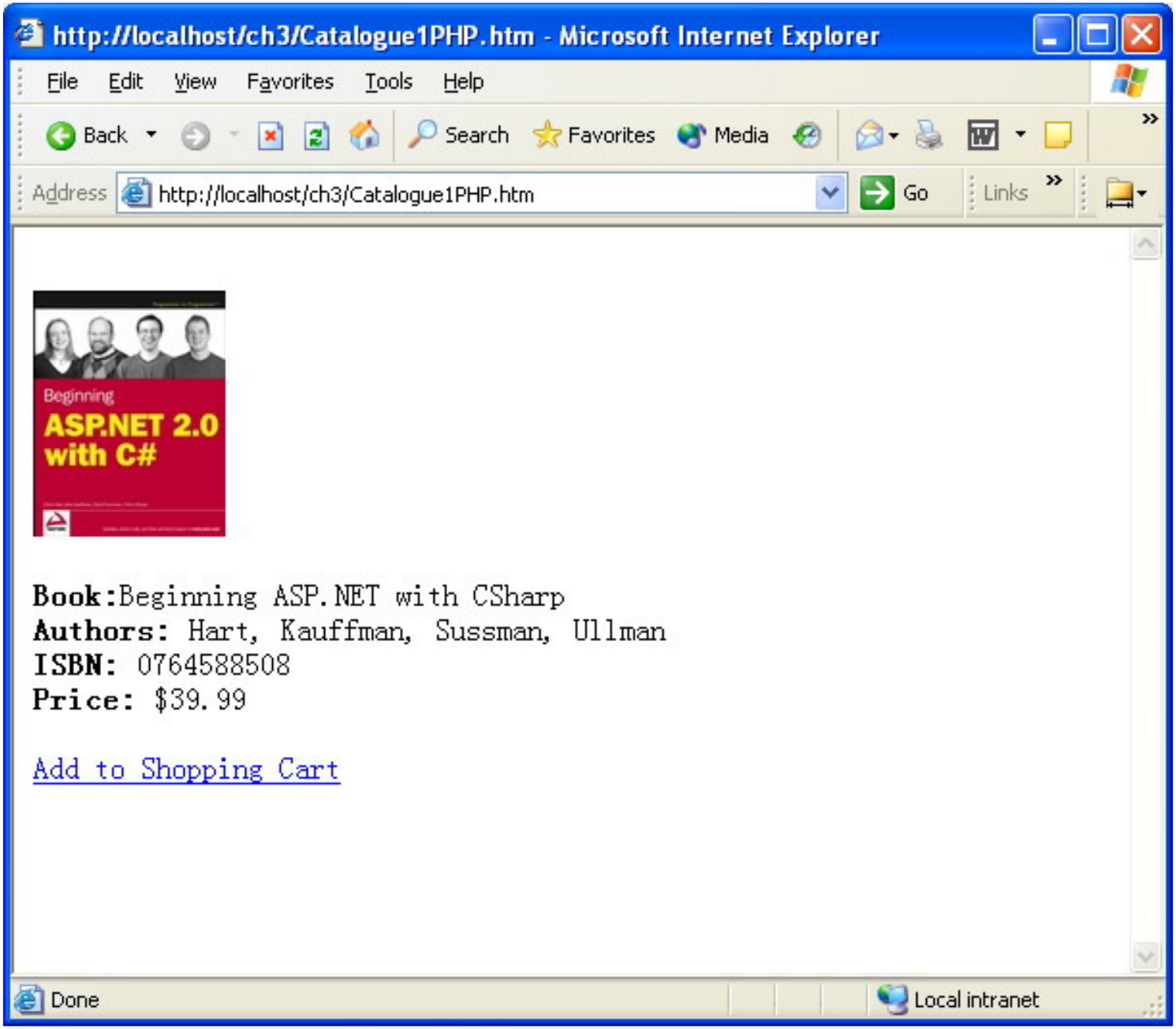
Clicking Add to Shopping Cart



Clicking Add Two More Times



Clicking Remove Item



Maintaining the Shopping Cart

- The shopping cart has to be stored as state information between user interactions
- PHP superglobal $_SESSION discussed in Lecture 6 can be used to keep the state information

Maintaining the Shopping Cart

- As mentioned, we choose to maintain the shopping cart in the session
- For communication between server and client we represent the cart as an XML document

```
<?xml version="1.0"?>
<cart>
  <book>
    <title>Book Title</title>
    <quantity> 3 </quantity>
  </book>
</cart>
```

If there were more books in the catalogue, then there may be additional book items in the XML

- The cart has to be updated following each user-interaction
- We are adding just one book element, or removing it, or changing the quantity, but we should write code that can be readily modified to handle multiple books

Client Page – Catalogue1PHP.htm

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
   <script type="text/javascript" src="xhr.js"></script>
   <script type="text/javascript" src="CartPHP.js"></script>
</head>
<body>
   <br/><img id="cover" src="begaspnet.jpg" /><br /><br />
   <b>Book:</b><span id="book">Beginning ASP.NET with CSharp</span><br />
   <b>Authors: </b><span id="authors"> Hart, Kauffman, Sussman, Ullman</span>
   <br /><b>ISBN: </b><span id="ISBN">0764588508</span>
   <br /><b>Price: </b><span id="price">$39.99</span>
   <br /><br />
   <a href="#" onclick="AddToCart()" >Add to Shopping Cart</a>
   <br /><br />
   <span id="cart" ></span>
</body>
</html>
```

Notes

- Note that we are storing the details of the book in the client (hard-coded)
- This is not suitable for a real application, where the book catalogue would be stored on the server, in a database or in XML form, with suitable identifying details retrieved by the client on loading, and then displayed in the client document, to enable the user to select books for purchase

Representing the Cart

- On the server : an associative array, that pairs the book title with the number of copies in the cart
- As data for transferring between server and client : as an XML document, to be sent from server as text, but picked up on the client (in the responseXML property of an XHR object) as an XML DOM object
- On the client : an XML DOM object, that can be read, and changed

CartPHP.js

```
var xhr = createRequest(); // from xhr.js (in Lecture 1)

function AddToCart()
{   var book  = document.getElementById("book").innerHTML;

    xhr.open("GET", "ManageCart.php?action=Add&book=" +
        encodeURIComponent(book) + "&value=" + Number(new Date), true);
    xhr.onreadystatechange = getData;
    xhr.send(null);
}

function DeleteFromCart()
{   var book  = document.getElementById("book").innerHTML;

    xhr.open("GET", "ManageCart.php?action=Remove&book=" +
        encodeURIComponent(book) + "&value=" + Number(new Date), true);
    xhr.onreadystatechange = getData;
    xhr.send(null);
}
```

Here the variable "book" stores the name of the single element with id "book" stored in the html document. If there are more books, this code will need to be altered

Number(new Date) is to force URL to be different from last time, so that IE aggressive cache doesn't occur

The action "add" and the book name are passed to the server, so that the shopping cart can be altered there.

The PHP function in ManageCart.php will use the value of the action parameter to determine what processing to do

CartPHP.js

```
function getData()
{   if ((xhr.readyState == 4) &&(xhr.status == 200))
    {   var serverResponse = xhr.responseXML;
        var cartDisplay = document.getElementById("cart");
        if (serverResponse == null){cartDisplay.innerHTML = "";}
        else{
           var books = serverResponse.getElementsByTagName("book");
            cartDisplay.innerHTML = "";
        for (i=0; i<books.length; i++) // this will handle any number of books in the cart
       {  if (window.ActiveXObject) // IE uses "text"
          { cartDisplay.innerHTML += " " +books[i].firstChild.text;
            cartDisplay.innerHTML += " " + books[i].lastChild.text + " " + "<a href='#'
            onclick='DeleteFromCart'>Remove Item</a>";
          }
          else
          { cartDisplay.innerHTML += " " +books[i].firstChild.textContent; // WC3 uses textContent
            cartDisplay.innerHTML += " " + books[i].lastChild.textContent + " " + "<a
            href="#" onclick="DeleteFromCart()">Remove Item</a>";
     } } } } }
```

Here, by contrast, the variable "books" stores the array of book elements stored in the shopping cart, sent from the server

The firstChild of a book element gives the name of the book.

Notes

1. To access the text that represents the book name, the text stored in the first child node of the book has to be used. In old IE this means using **text**, whereas in WC3-compliant browsers, textContent is used. (Note that IE8 does not have textContent. We need conditional code. (Eventually when everything is compliant, we can get rid of this nuisance!!))
2. When using the GET protocol, IE caches the page, and so the data passed will be the same as last time, unless the query string is changed. We force a change of query string by appending "&value=" + Number(new Date). Since the Date will be different on each call, the URL used in the GET will be different, and so the cached call will not be used.
3. An alternative is to use the POST protocol that would be a better approach, and you should write that alternative code.

ManageCart.php

```
<?php
   session_start(); // start a session
    header('Content-Type: text/xml');
?>
<?php
    $newitem = $_GET["book"]; // book name
    $action = $_GET["action"];  // add or remove?
//    if ($_SESSION["Cart"] != "") // this line is obsolete in new version of PHP engine
     if (array_key_exists("Cart", $_SESSION)) // the “cart” already exists with an item in it
    {
        $myCart = $_SESSION["Cart"]; // assign the session variable to $myCart
        if ($action == "Add") // we are processing an “Add” request
        {
            if ($myCart[$newitem] != "") // the cart already has this book in it
            {
                $value = $myCart[$newitem] + 1; $myCart[$newitem] = $value; // add 1 to no of copies
             }
            else // this is the first copy of this book to be added (there may be other books)
```

ManageCart.php (Cont'd)

```
        else // this is the first copy of this book to be added (there may be other books)
        {
            $myCart[$newitem] = "1";
        }
    }
    else // we are processing a "Remove" request
    {
        $myCart= ""; // nb – we are clearing the whole cart; this is what the spec requires
    }
}
else // the "cart" is NOT present – ie, we have no books ordered; order one!
{
    $myCart[$newitem] = "1";
}
// copy modified cart to session variable, convert to serialized XML and send to client
$_SESSION["Cart"] = $myCart;
echo (toXml($myCart)); // function toXML
```

Notes

1. We first register this session, and give it a name “Cart”
2. We also set the type of information to be returned (text/xml)
3. We then pick up the information passed from the client – the book details and whether we are adding or removing an item
4. And then deal with the following cases :
    1. The cart is currently not empty
        1. We are adding an item
            1. The cart already has THIS book in it
            2. The cart does not already have THIS book in it
        2. We are removing an item
            - Note that here we just clear the whole cart (if spec requires that we just cut one copy, different code is needed)
    2. The cart is currently empty

Notes

5. We deal partly with the possibility that the book catalogue has more than one book; we correctly add a book (whether another copy of a book in the cart, or a new book not yet in the cart), but when we delete an item, the code currently deletes the whole cart. This is not right if the cart contains more than one book.
6. We conclude by saving the changed cart back to the session variable (to be picked up on next user-interaction), and then convert the cart to XML, serialize this as a string, and send to the XHR object

ManageCart.php (Cont'd)

```
function toXml($aCart)
{   $doc = new DomDocument('1.0');
    $cart = $doc->createElement('cart');
    $doc->appendChild($cart);
    foreach ($aCart as $Item => $ItemName)
    {   $book = $doc->createElement('book');
        $cart->appendChild($book);
        $title = $doc->createElement('title');
        $book->appendChild($title);
        $value = $doc->createTextNode($Item);
        $title->appendChild($value);
        $quantity = $doc->createElement('quantity');
        $book->appendChild($quantity);
        $value2 = $doc->createTextNode($ItemName);
        $quantity->appendChild($value2);
    }
    $strXml = $doc->saveXML(); // this serializes the XML as a string
    return $strXml;
  }
?>
```

This is how to loop through the items of an associative array

```
<?xml version="1.0"?>
<cart>
  <book>
    <title>Book Title</title>
    <quantity> 3 </quantity>
  </book>
</cart>
```