Midterm ch.6-9

> Some basic concepts may ignored.

Document Object Model

- What
  - Document Object Model (DOM) is a programming interface for XML documents
  - The XML DOM is designed to be used with **any programming language** and any operating system.
- Functions

  `document` is root, `.getElementById`, `.getElementsByTagName`, `obj.innerHTML`, `obj.style.left`...
  - innerHTML
    1. Elements that do not have both an opening and closing tag cannot have an innerHTML property.
    2. When the innerHTML property is set, the given string completely **replaces** the existing content of the object.
    3. don't use the +=, Every time innerHTML is set, the HTML has to be parsed, a DOM constructed, and inserted into the document. This takes time.

       -> call appendChild

       ```
       var newElement = document.createElement(’div');
       newElement.innerHTML = '<p>Hello World!</p>';
       elm.appendChild(newElement);
       //This way, the existing contents of elm are not parsed again.
       ```
- Using a DOM Parser with Javascript
  1. Microsoft XML parser, IE: `var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")`
  2. Netscape-based browsers (Firefox): `var xmlDoc= document.implementation.createDocument("","doc",null);`
  3. Newer browsers use “Synchronous” **XMLHttpRequest**
     - Update a web page without reloading the page
     - Request data from a server after page has loaded
     - Receive data from a server after page has loaded
     - Send data to a server in the background

  ```
  var xmlDoc;
  function loadXML(url) {
  ```

```
    if (window.XMLHttpRequest){// code for IE7+, Firefox, Chrome, Opera, Safari
        xmlhttp=new XMLHttpRequest();
    }
    else {// code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.open("GET",url,false); // ‘false’ = synchronous request
    xmlhttp.send(); // open, send, responseXML are
    xmlDoc=xmlhttp.responseXML; // properties of XMLHttpRequest
    return xmlDoc; // (file returned in responseXML
} // or responseText)
// ....... processing the document goes here
```

4. XML
    - Node Type

      1 ELEMENT_NODE, 2 ATTRIBUTE_NODE, 3 TEXT_NODE, 4 CDATA_SECTION_NODE, 5 ENTITY_REFERENCE_NODE, 6 ENTITY_NODE, 7 PROCESSING_INSTRUCTION_NODE, 8 COMMENT_NODE, 9 DOCUMENT_NODE, 10 DOCUMENT_TYPE_NODE, 11 DOCUMENT_FRAGMENT_NODE
    - example

```
- <bookstore>
    <book category="cooking">
      <title lang="en">Everyday Italian</title>
      <author>Giada De Laurentiis</author>
      <year>2005</year>
      <price>30.00</price>
    </book>
+ <book>
+ <book>
</bookstore>
<!--
1. ELEMENT_NODE (type 1): bookstore, book, title, author, year, price
2. TEXT_NODE (type 3): "/n" nodes, "Everyday Italian", "30.00", …
3. Hint: element nodes have children, text nodes are leaves
4. x[i].nodeType == 1: tests for element nodes, text nodes (like "\n" ) are ignored
-->
```

- Examples

1. `document.getElementById(id).AttName = Value;`
2. `this.getAttribute('AttName');`
3. `this.setAttribute('AttName','Value');`

4.
```
// An Example
<sentence> The &projectName; <![CDATA[<i>project</i>]]> is <?
editor: red><bold>important</bold><?editor: normal>.
</sentence>

// DOM structure
+ ELEMENT: sentence
    + TEXT: The
    + ENTITY REF: projectName   // EntityReference
        + COMMENT: The latest name we're using
        + TEXT: Eagle
    + CDATA: <i>project</i>     // CDATA section
    + TEXT: is
    + PI: editor: red   //  processing instructions <?...?>
        + ELEMENT: bold
    + TEXT: important
    + PI: editor: normal
```

- Summary of XML/HTML node types and children
    1. Document -- Element(maximum of one), ProcessingInstruction, Comment, DocumentType (maximum of one)
    2. DocumentFragment -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
    3. DocumentType -- no children
    4. EntityReference -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
    5. Element -- Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
    6. Attr -- Text, EntityReference
    7. ProcessingInstruction -- no children
    8. Comment -- no children
    9. Text -- no children
    10. CDATASection -- no children
    11. Entity -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
    12. Notation -- no children
- 3 Different Solutions and Observations
    1. It works on IE7, IE8, IE9 and Firefox but not on Chrome.
    2. Uses bookstore.childNodes and XMLHttpRequest
    3. [bookstore.children, not work in IE, because children is a DOM Level 4 property,

       and children is different in IE](http://cs-server.usc.edu:45678/examples/dom/example2.html)

Forms and Common Gateway Interface Mechanism

- Forms
    - Introduced in HTML 2.0
    - 1. Use HTML form elements to create the page
      2. Write a server-side script to process form data
- Some Attributes
    1. ACTION=URI (form handler)
    2. METHOD=[get | post] (HTTP method for submitting form)

       GET is the default; form contents are appended to the URL

       POST causes the fill-out form contents to be sent in a data body as standard input
    3. ENCTYPE=ContentType (content type to submit form as)

       Defaults to **application/x-www-urlencoded** which returns name/value pairs, separated by &, spaces replaced by + and reserved characters (like #) replaced by %HH, H a hex digit
    4. ACCEPT-CHARSET=Charsets (supported character encodings)
    5. TARGET=FrameTarget (frame to render form result in, in HTML4)

       (a browsing context name or keyword, in HTML5, such as *self,* blank, _parent, _top, iframename)
    6. ONSUBMIT=Script (form was submitted)
    7. ONRESET=Script (form was reset)
- `<INPUT>`
    - TYPE: [CHECKBOX | FILE | HIDDEN (The field is not rendered, so servers can maintain state information) | IMAGE (graphical submit buttons) | PASSWORD(input is echoed with *) | RADIO (take a single value from a set of alternatives; all buttons have same name and explicit value) | RESET | SUBMI | TEXT]

      [COLOR | DATE(date) | DATETIME(date and time(with time zone)) | DATETIME-LOCAL(date and time (no time zone)) | EMAIL | MONTH(month/year) | NUMBER(w/wo min="1" max="5") | RANGE | SEARCH|TEL|TIME|URL|WEEK(week and year)]
    - NAME, VALUE, CHECKED, DISABLED, READONLY
    - eg. `<FORM METHOD="POST" ACTION="/cgi-bin/post-query">`

      Note:post-query is a standard Apache CGI program distributed by web servers and used to check that form elements are being properly sent to the server
    - `<TEXTAREA NAME="narrowarea" ROWS=1 COLS=40>This is 1 x 40</TEXTAREA>`
    - `<select size=1 required><option selected></option><option disabled></option></select>`
    - `<fieldset><legend accesskey=""></legend>...</fieldset>`

      LEGEND to provide a caption for the group of controls

- Common Gateway Interface (CGI)
  - scripts -> create dynamic Web documents
  - 1. Scripts are placed in a server directory often named cgi-bin
    2. Scripts can deliver information that is not directly readable by clients
    3. Scripts dynamically convert data from a non- Web source (e.g. DBMS) into a Web-compatible document
  - `Web browser` —[send Query via URL or stdin] —> `Web server` —[interpret -> invoke CGI script] —> `Script in Gateway` (may need Database) —[return output] —> `Web server` —[may return HTML]—> `Web browser`
  - Invloke CGI Script, eg. `<a href="http://domain_name/cgi-bin/scriptname?arg1+arg2">`
- Languages
  - compiled languages: C/C++
  - **interpreted** languages: PHP, JavaScript or Java
- CGI Script Environment Variables
  - a set of pre-defined dynamic values
  - created by the web server and set immediately before the web server executes a gateway script
  - can retrieve the values and use the data they send
  - 1. Non-request specific (same for all requests)
       - `SERVER_SOFTWARE = Apache/1.3.15`, the name and version of the information server software answering the request. SERVER_SOFTWARE = Apache/1.3.15
       - `SERVER_NAME = nunki.usc.edu`, server's hostname, DNS alias, or IP address
       - `GATEWAY_INTERFACE`, the revision of the CGI specification with which this server complies
       - `SERVER_PROTOCOL = HTTP/1.0`, the name and revision of the information protocol with which this request came in
       - `SERVER_PORT = 8088`, the port number to which the request was sent
    2. Request specific (set depending on each request)
       - `REQUEST_METHOD`, the method with which the request was made; e.g., (GET, POST)
       - `PATH_INFO`, the extra path information as given by the client; e.g., given http://nunki.usc.edu:8080/cgi-bin/test.cgi/extra/path then `PATH_INFO = /extra/path`
       - `PATH_TRANSLATED`, the `PATH_INFO` path translated into an absolute document path on the local system `PATH_TRANSLATED = /auto/home-scf-03/csci571/WebServer/apache_1.2.5/htdocs/extra/path`
       - `SCRIPT_NAME = /cgi-bin/test.cgi`, the path and name of the script being accessed as referenced in the URL

            - `QUERY_STRING`, the information that follows the ? in the URL that referenced this script
            - `REMOTE_HOST`, `REMOTE_ADDR`, `AUTH_TYPE`, `REMOTE_USER`, `REMOTE_IDENT`, `CONTENT_TYPE`, `CONTENT_LENGTH`
            - `HTTP_headerX` contains the refuest header X field data, eg. `HTTP_USER_AGENT = Mozilla/4.7 [en]C-DIAL (WinNT; U)`, `HTTP_ACCEPT`, `HTTP_REFERER`

- Output
    - document(HTML, plain text, image, video or audio clip...), Instructions, an error indicator
    - Output begins with header. Any headers that are not server directives are sent directly back to the client
    - Server Directives: to inform the server about the type of output
        1. Content-type: type/subtype

           The MIME type of the document being returned, eg. `text/html`(HTML document), `text/plain` (plain-text document)
        2. Location: Alerts the server that the script is returning a reference to a document, not an actual document

           eg. `location: http://www.ncsa.uiuc.edu/`, If the argument is a URL, the server will issue a redirect to the client;

           eg. `location: /path/doc.txt`, a path, the document specified will be retrieved by the server, starting at the document root
        3. Status: give the server an HTTP/1.1 status line to send to the client. `nnn(three-digit status code) xxxx(informative message)`
- Check things to be readable and excutable by the server
    - CGI scripts, Other programs that the scripts call, The directory in which the scripts reside
    - In UNIX, check the **read/write/execute permissions** of the files and directories
    - In Windows, check **the web server settings** of the script directories
- PHP
    - with built-in ability to access environment variables
    - show_vars.php -> prints environment variables

JSON – JavaScript Object Notation

- What
    - a lightweight data interchange format

      It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects).

    - MIME type: application/json, extension: .json
    - The JSON format is often used for transmitting structured data over a network connection in a process called serialization.
- How
    1. Client side (browser) — browser processing: JSON file/data -> variable -> object
    2. Server side — server processing: parsers process it and may convert -> classes and attribute of the language, eg. PHP, Java
    3. Data exchange — between them
        - Loading a JSON file from the server:
            1. directly including the file into the HTML page, as a JavaScript .json external file.
            2. loading by a JavaScript command
            3. using XMLHttpRequest
        - convert JSON into an object can via JavaScript `eval()`
        - sending the file to the server may via `XMLHttpRequest`
    4. example

```
//The XMLHttpRequest code:
var req = new XMLHttpRequest();
req.open("GET", "file.json", true);
req.onreadystatechange = myCode; // the callback
req.send(null);

//The JavaScript callback: eval() parses JSON, creates an object and assigns it to variable doc
function myCode() {
if (req.readyState == 4) {
    if (req.Status == 200) {
        var doc = eval('(' + req.responseText + ')');
    }
}

//Using the data:
var menuName = doc.getElementById('menu'); // finding a field menu
doc.menu.value = "my name is"; // assigning a value to the field

//How to access data:
doc.commands[0].title // read value of the "title" field in the array
doc.commands[0].action // read value of the "action" field in the array
```

- `Eval()`: is subject to security vulnerabilities if the data and the entire JavaScript environment is not within the control of a single trusted source;
- JSON Basic Data Types

  String (**double**-quoted unicode with backslash escaping); Numbers (integer, real, or floating point); Booleans; Object {"key":value,...}; Array [], can start array indexing at 0 or 1; Null
- JSON is Not XML

  JSON: Objects, Arrays, Strings, Numbers, Booleans, null

  XML: element, attribute, Attribute string, content, <![CDATA[ ]]>, Entities, Declarations, Schema, Stylesheets, Comments, Version, namespace
- JSON Parsers
  1. decoder must accept all well-formed JSON text, may also accept non-JSON text
  2. encoder must only produce well-formed JSON text
- Same Origin Policy
  - a security feature that browsers apply to client-side scripts

    Same protocal & host & port
- The Cross-Domain Hack
  - `<script src=http://otherdomain.com/data.js> </script>`

    that JSON will become a global variable in the webpage

    -> So JSON can be used to grab data from other servers, without the use of a server-side proxy
- JSON and Dynamic Script Tag "Hack"
  - Let data can only come from a single domain

    `this.noCacheIE = '&noCacheIE=' + (new Date()).getTime();`

    `this.scriptObj.setAttribute("src", this.fullUrl + this.noCacheIE);`, which sets the src attribute of the