

DOM Manipulation and Client-side Views

By: Haitham Mohammad



- **Document Object Model**

- × What is DOM?
- × DOM manipulation

- **Client-side Views**

- × Traditional Ajax model
 - ◆ Introducing Prototype Ajax
- × Client-side views using DOM
 - ◆ Introducing JSON
- × Moving views to client side
 - ◆ Introducing JavaScript Templates

Document Object Model

What is DOM?



- **Document Object Model** is a standard object model for representing HTML, XML and related formats.
- The Document Object Model is
 - × Platform independent
 - × Language independent
- DOM is used by web browsers to maintain a buffered structure that corresponds to the rendered HTML document hierarchy.
- Changes committed to DOM is automatically reflected to the rendered HTML, and vice-versa.

Example:

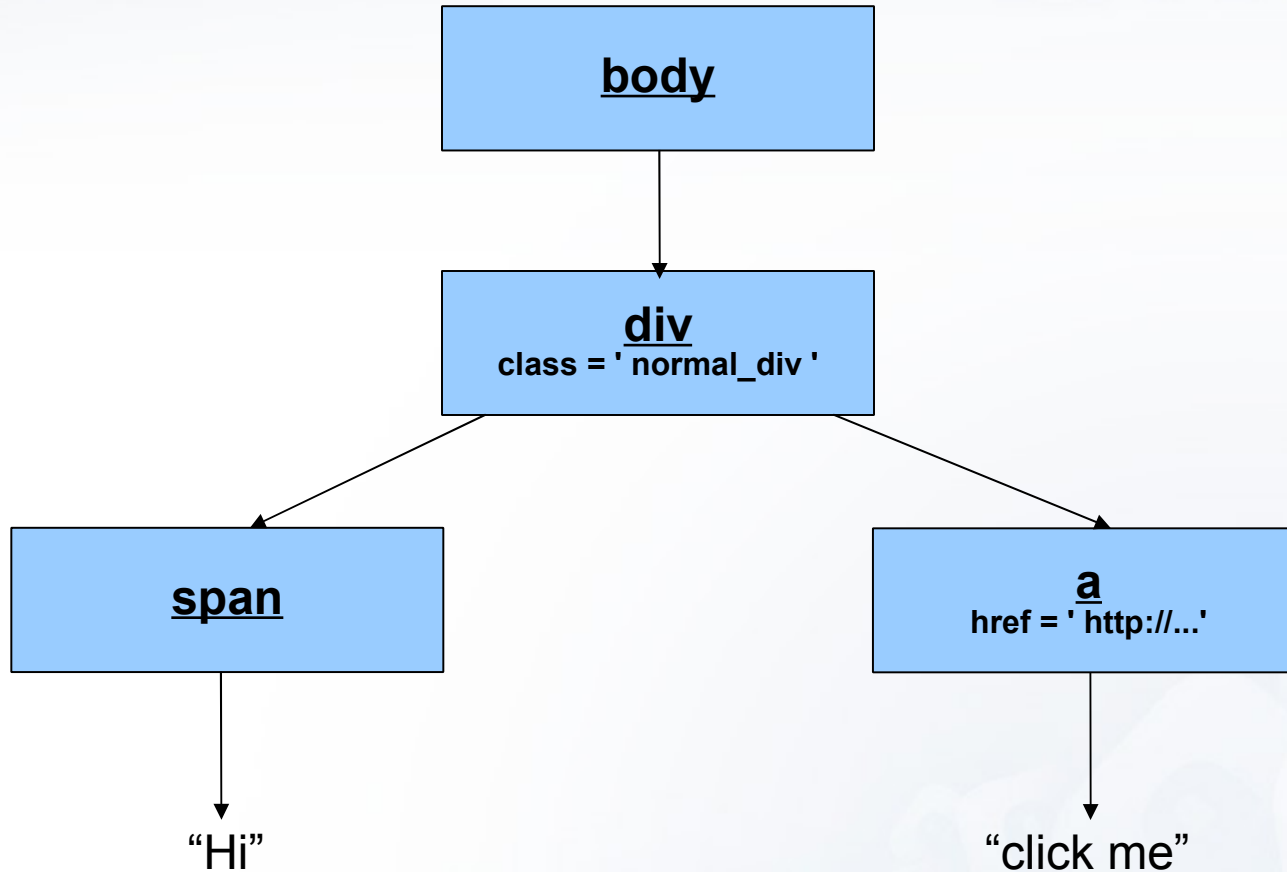


HTML:

```
<html>
  <head>...</head>
  <body>
    <div class='normal_div'>
      <span>Hi</span>
      <a href='http://www.haitham.com'>click me</a>
    </div>
  </body>
</html>
```

Example:

DOM:



- Traversal

```
* node = document.getElementById('node_id');  
* children = node.childNodes;  
* parent = node.parentNode;  
* sibling = node.nextSibling;  
* sibling = node.previousSibling;  
* first = node.firstChild;  
* last = node.lastChild;
```

- Add

```
* node = document.getElementById('node_id');  
* newNode = document.createElement('span');  
* node.insertBefore(newNode, node.firstChild);  
* node.appendChild(newNode);  
  
* table = document.getElementById('table_id');  
* table.insertRow(3);  
* table.childNodes[4].insertCell(5);
```

- Edit

```
* node = document.getElementById('node_id');  
* node.id = 'node_id2';  
* node.className = 'node_class'  
* node.disabled = true;  
* node.onclick = function(event){....};  
* node.style.marginLeft = '20px';  
* node.innerHTML = "<span>hi</span>"
```

- Delete

```
* node = document.getElementById('node_id');  
* child = node.childNodes[3];  
* node.removeChild(child);  
* node.replaceChild(newChild, oldChild);
```

Client-side Views



Traditional Ajax Model



- Client creates a new xhr.
- Client sets the xhr's success callback.
- Client sends the request.
- Server evaluates some view template using runtime data.
- Server responds with the evaluated HTML.
- The xhr's success callback typically updates an element in the document with the response html.

Traditional Ajax Model



```
function handler()
{
    if(this.readyState == 4 && this.status == 200)
        document.getElementById('id').innerHTML = this.responseText;
    else if (this.readyState == 4 && this.status != 200)
        error();
}

var client = new XMLHttpRequest();
client.onreadystatechange = handler;
client.open("GET", url);
client.send();
```

Prototype Ajax:



```
New Ajax.Request( url, { method: 'GET',  
    onSuccess = function(transport){  
        $('id').innerHTML = transport.responseText;  
    } } );
```

- * **Ajax.Updater**
- * **Ajax.PeriodicalUpdater**
- * **Ajax.Responders**

<http://www.prototypejs.org/api/ajax>

A Simple Example



- Suppose we need to retrieve an index of all posts in a blog.
- The client sends an Ajax request to '/posts'.
- The server retrieves all posts and evaluates a template to produce an HTML snippet of all-posts index.
- The server responds with the evaluated HTML.
- The client Ajax onSuccess callback updates a place holder with the response.
- Our ultimate goal is to move the views description and logic to the client.

A Simple Example



- Client:

```
new Ajax.Request( "/posts", {method: 'GET',  
    onSuccess: function(transport){  
        $('posts_place_holder').innerHTML = transport.responseText;  
    } } );
```

A Simple Example



- Server (in RoR for example) :

```
@user = current_user
```

```
@posts = all_posts
```

```
<%for post in @posts%>
```

```
<div class='post_div'>
```

```
  <a href='<%= "/posts/#{post.id}" %>'> <%=post.title%> </a>
```

```
  <%if @user.is_admin?%>
```

```
  <a href="javascript:deletePost(<%=post.id%>)">delete</a>
```

```
  <%end%>
```

```
</div>
```

```
<%end%>
```

Client-side Views (DOM)



- First modification level
- The server responds with some standard data format
- The client receives the retrieved data from the server
- The client uses this data to apply some DOM manipulation to update the document
- A perfectly suitable data format to use in this situation is JSON

Client-side Views (DOM)



- JSON

```
dataObject = {'id': 1, 'name': "haitham"};
```

equivalent to:

```
dataObject = eval( (" {\\"id\\": 1, \\"name\\": \\"haitham\\"} ") );
```

The server application serializes objects into JSON format which can be evaluated into JS objects in only one line.

Client-side Views (DOM)



- **Server (modified):**

```
render :text => { :user => @user, :posts => @posts }.to_json
```

- **Client (modified):**

```
new Ajax.Request( "/posts", {method: 'GET',  
  onSuccess: function(transport){  
    data = eval( '(' + transport.responseText + ')' );  
    updatePosts(data);  
  } } );
```

Client-side Views (DOM)



```
function updatePosts(data){
  ph = $('posts_place_holder');
  for (var i=0; i<data.posts.length; i++){
    div = document.createElement('div');
    div.className = 'posts_div';
    a1 = document.createElement('a');
    a1.href = "/posts/" + data.posts[i].id;
    a1.innerHTML = data.posts[i].title;
    div.appendChild(a1);
    if ( data.user.is_admin ){
      a2 = document.createElement('a');
      a2.href = "javascript:deletePost(" + data.posts[i].id + ")";
      a2.innerHTML = "delete";
      div.appendChild(a1);
    }
    ph.appendChild(div);
  }
}
```

Client-side Views (JST)



- Second modification level
- The server responds with some standard data format (JSON)
- The client receives the retrieved data from the server
- The client uses this data to evaluate a JavaScript Template.
- The html place holder is updated with the result.

Client-side Views (JST)



- JavaScript Template

```
<textarea id='jst_posts' style='display:none;'>
  {for post in posts}
  <div class='post_div'>
    <a href='/posts/${post.id}'> ${post.title} </a>
    {if user.is_admin}
    <a href='javascript:deletePost(${post.id})'>delete</a>
    {/if}
  </div>
  {/for}
</textarea>
```

Client-side Views (JST)



- Client (modified):

```
function updatePosts(data) {  
  postsHTML = TrimPath.processDOMTemplate('jst_posts', data);  
  $('posts_place_holder').innerHTML = postsHTML;  
}
```

Advantages



- Natural support for the web services paradigm
- Easier development of multi-client applications
- Bandwidth optimization
- Response-time optimization

References



- <http://www.prototypejs.org>
- <http://www.josn.org>
- <http://code.google.com/p/trimpath/wiki/JavaScriptTemplates>

Thank You

e space[®]

