Webbprogrammering, DA123A
Malmö University, School of Technology
Andreas Göransson, 2011-04-05

Introduction

This document explains, both through visualization and detailed instructions through code, how to manipulate a parsed HTML document in order to create a dynamic experience in an otherwise static application.

I believe one of the most common issues for students, when introduced to the concept of dynamic websites, is the idea of the HTML document as *the website* when in-fact the HTML document is just one of many pieces in the puzzle of a dynamic website. Let's first have a look at the HTML document, and what it is.

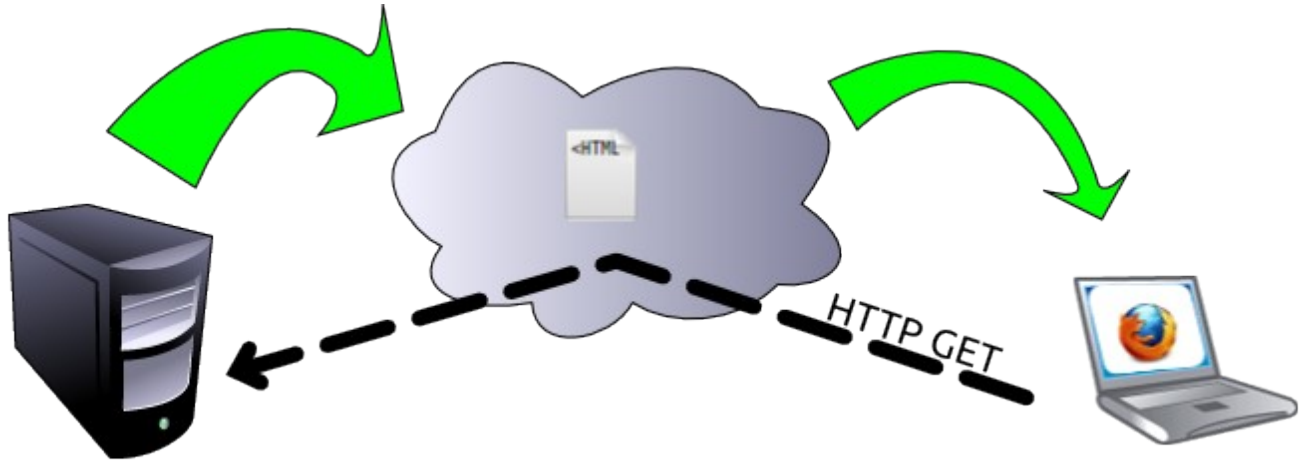

Illustration 1: server - cloud - client

The HTML document

y'know... the index.html thing you've written so many times by now...

So, the HTML document (*.html, *.htm) is a static XML-based document. That means the HTML document is really firm in the way it exists. When you've written a document and publish it to a website it doesn't change shape without you re-programming it – that is, for the document to change you need to change it and then save it and upload once more to the web server.

However, there's something we can do to actively change the contents of a HTML document as viewed by the user (the dude/dudette with the web browser) and that's called JavaScript. Hold on, let's back up a step...

The Document (Object Model)

The document object model (or DOM for short) is the parsed version of the HTML document which the web server sends to the users web browser when receiving a HTTP GET request, and it exists inside the users browser – for the sake of simplicity, let's view the DOM as the compiled code of our

web application, the *.exe of our web site[1] – and just like any other application our website contains certain variables.

Some of these variables contain the current view of the our web site – as seen by the users web browser – and by changing the content of these specific variables we can change the way our website is rendered by the client (the client is in this context the users web browser).

An important thing to remember though! We're never changing the HTML documents on the server, those documents remain static and unchanged! The only thing we're actually changing through Javascript is the way our users perceive the website, and we do this by modifying the Document Object Model (DOM) which is contained inside our users browser.

Bear with me, we'll get to the code shortly... but first, how do we modify the DOM?

Manipulating the Document Object Model

Consider the DOM a tree set firmly in the ground, and as we traverse the whole tree we pass different tags on the way. At the root of the tree we've got the <html> tag!

Consider yourself the master gardener, you possess the amazing ability to instantly grow branches on the tree where you please! This is in essence what JavaScript allows us to do, we can insert new elements inside the Document Object Model where we see fit, or change/delete the already present elements in the tree.

The result can be a completely different view of our html document. Actually, the HTML documents <body> tag can actually be completely void of content, while still looking like a normal website to the user.

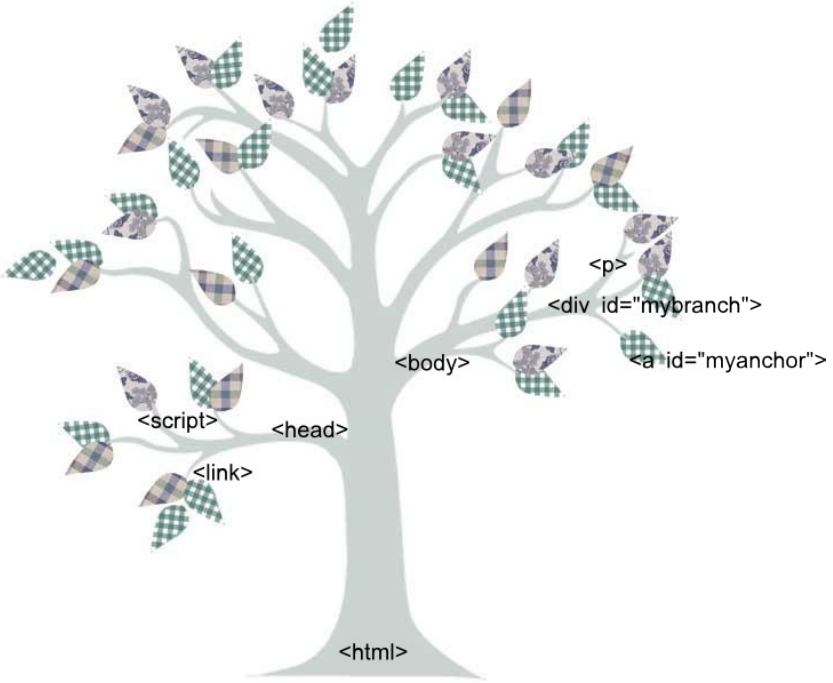

Illustration 2: The DOM tree, concept

1 Important! The parsed information is not compiled code, it's actually interpreted code.lullhsusj

Note, you can also view the actual DOM tree by using plug-ins such as the FireFox DOM inspector – seen below.

In the background you can see a very simple HTML document, viewed in FireFox, in the foreground is the DOM Inspector showing the *tree* for the HTML document.

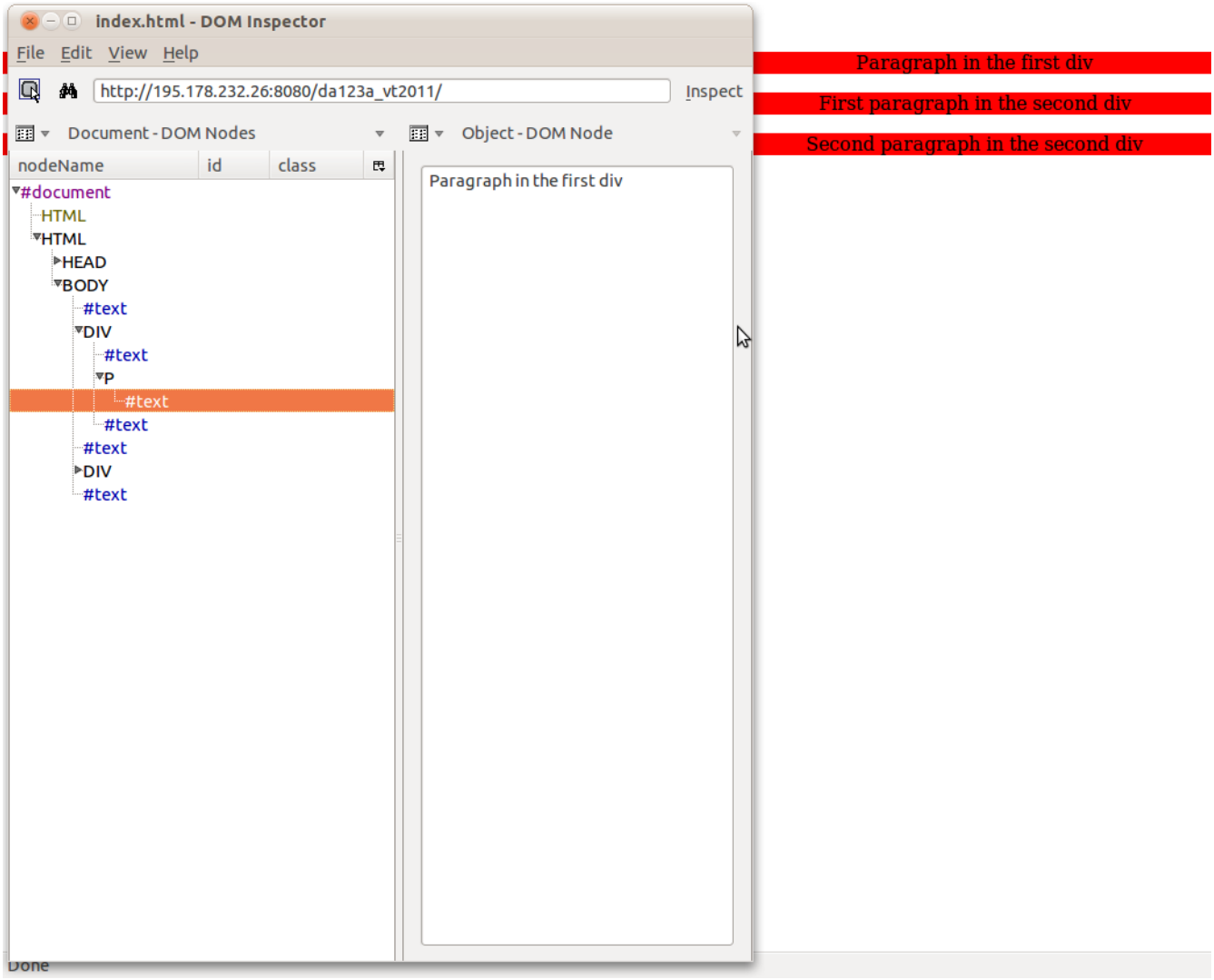

Illustration 3: DOM Inspector, available in FireFox

Finally, the code!

Under this topic I'll present how to use specific JavaScript functions which allows you to manipulate the Document Object Model of any website, and I'll do so through using the previous example – *the DOM tree.*

Attached at the end of the document are the three files used when writing this document.

Almost, first let's mention some important objects within the DOM

Within the DOM exists some objects which let's us read and manipulate the current view of the users browser. The two most important (and also the ones we'll focus on in this short tutorial) are the

document and the *window* objects.

If we're going to stay true to our previous analogy, we should mention that in this sense the *document* object is the tree stem, and that the *window* object would be the ground in which it grows.

Read more about these, and others, on http://www.w3schools.com/

Example: Cutting a branch off the tree (Delete element from the DOM)

If we simply want to delete an already existing branch of the tree, we'll employ the rather brute method of a hack saw.

Consider the tree we viewed before, and that we want to cut the anchor-leaf off of that tree! In this example we'll use a targeted removal, that means we're targeting a specific element by unique identifiers. We could also use more generic targeting methods such as removing the **first child** or **last child**, or even remove the **current element**.

For more information on removal, see: http://www.w3schools.com/dom/dom_nodes_remove.asp

Back to our example – targeted removal of DOM node.

First we need to locate the branch which the leaf is growing on, this can be done in several ways, but we'll use the popular function getElementById – as seen below.

```
// Get the parent element
var branchelement = document.getElementById( “mybranch” );
```

Now that we have the branch where the anchor-leaf is growing, we need to identify the anchor-leaf – again we'll use the getElementById funtions.

```
// Get the anchor-leaf element
var anchorelement = document.getElementById( “myanchor” );
```

Now that we know both our target element (the one that we want to remove) and it's parent (where to remove it from) we can cut the leaf off it's parent branch.

```
// Delete the anchor-leaf from the parent branch
branchelement.removeChild( anchorelement );
```

Done! (Note, there are several ways of doing removals, you should investigate all of them in detail at w3schools.

Example: Trim a branch (Edit an already present element in the DOM)

Say our anchor-leaf is not edited properly inside the HTML document, it's missing the really important href attribute (y'know, the one that actually points to a new webpage). So, we'll use Javascript to add a proper http reference to that attribute.

Similar to the above example, we need to find the anchor-leaf we want to edit. We'll again use the getElementById method.

```
// Get the anchor-leaf element
var anchorelement = document.getElementById( “myanchor” );
```

Now that we've found the anchor-leaf we're free to add, remove, or change attributes of it – again there are several ways of doing these actions and I'll show you just one. Investigate w3schools for more

examples and a complete reference.

```
// Change the attribute of the anchorelement
anchorelement.setAtribute( “href”, “http://www.mah.se” );
```

That's all you need to edit attributes.

hint: adding a non-existent attribute is as simple as setting it!

Grow a new branch (Add a new element to the DOM)

Onwards to maybe the most difficult of all concepts, how do we add a new branch with new leafs on the DOM tree?

Again, to add a new element we need to know where to add it! There are several ways of finding a position to add the new branch – and we'll stick to the getElementById as we've done so far.

```
// Get the parent element
var branchelement = document.getElementById( “mybranch” );
```

When we've located the parent branch where we want to grow our new leaf (or branch...) we need to actually create the leaf – if we want to stay true to our analogy we should say that the new leaf is grown outside of the tree, and then attached to the tree at the correct position.

Let's create the new leaf.

```
// Create the new leaf
var newanchor = document.createElement( “a” );
// Populate the newanchor with the needed attribute
newanchor.setAttribute( “href”, “#” );
// Add the text to the newanchor
newanchor.appendChild( document.createTextNode(“new”) );
```

So, we've created a simple anchor-leaf, which we'll add to our parent branch. Now we need to know WHERE to place this new anchor-leaf on our parent branch. Again, we've several ways of doing this specific task. I'll show you two ways.

```
// Add the new anchor-leaf BEFORE the old anchor-leaf
branchelement.insertBefore( newanchor, oldanchor );
```

The second way of doing is is appending it to the very end of the parent branch element.

```
// Add the new anchor-leaf at the end of the parent branch
branchelement.appendChild( newanchor );
```

That's it! Not that difficult... was it?

index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

     <link rel="stylesheet" type="text/css" href="style.css">
     <script type="text/javascript" src="script.js"></script>

</head>

<body>

     <div id="mybranch">
          <a href="#" id="myanchor">myanchor</a>
     </div>

     <p>
          paragraph
     </p>

     <!-- This element is only used to invoke the javascript
functions!! -->
     <div>
          <a href="#" id="mybutton">Press me</a>
     </div>
</body>

</html>
```

style.css

```
body {
     margin:50px 0px;
     padding:0px;
     text-align:center;
}

p {
     background-color: #f00;
}

.myclass {
     background-color: #0f0;
}
```

script.js

```
function action(){
     /* Get parent branch */
     var parentbranch = document.getElementById( "mybranch" );
     var targetelement = document.getElementById( "myanchor" );

     /* Remove the targetelement from the parentbranch */
     parentbranch.removeChild( targetelement );

     /* Change the href attribute of the targetelement */
     //targetelement.setAttribute( "href", "http://www.mah.se" );
     //targetelement.setAttribute( "class", "myclass" );

     /* Create and add a new element to the DOM tree */
     //var newParagraph = document.createElement( "p" );
     /* Create the textnode for the paragraph, and append it to the
new paragraph */
     //newParagraph.appendChild( document.createTextNode("new
paragraph") );

     /* Add the new element to the parent branch - test both of
these, one at a time! */
     //parentbranch.insertBefore( newParagraph, targetelement );
     /* Or if we want to add it after an element... */
     //parentbranch.appendChild( newParagraph );
}

function initlisteners(){
     document.getElementById("mybutton").addEventListener( "click",
action, false );
}

window.addEventListener( "load", initlisteners, false );
```