IIT Madras

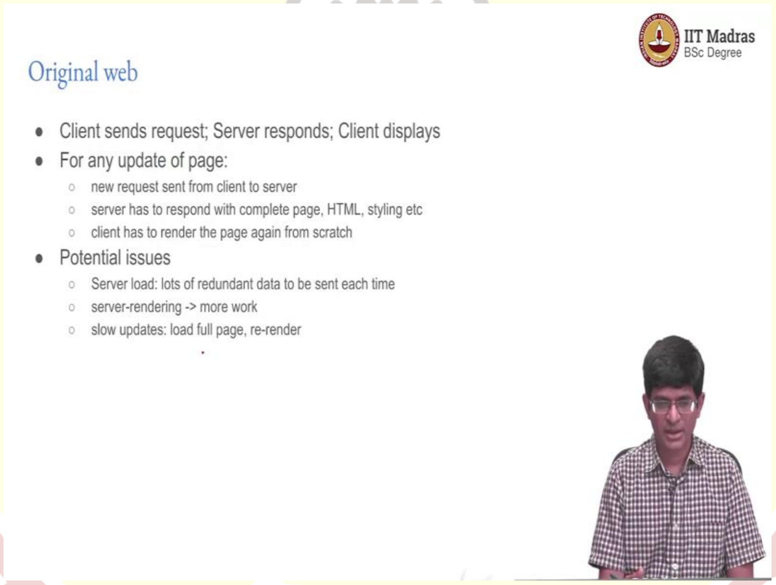
ONLINE DEGREE

Modern Application Development - I
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras

Asynchronous Updates

Hello everyone, and welcome to this course on Modern Application Development. So, so far, we have been looking at the different ways by which we can implement the frontend rendering, or at least in the generation of HTML pages. Either you could do it server side rendered, or you could have fully static, or you could have some kind of a mix and, do a certain amount of work on the client side. An important part of what makes that possible is this notion of Asynchronous Updates. So, let us try and understand what that is.

(Refer Slide Time: 00:42)



Original web

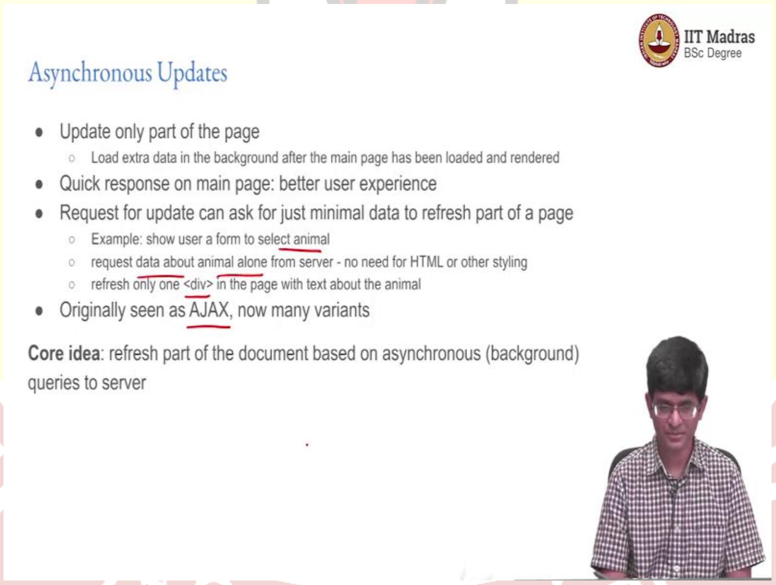
- Client sends request; Server responds; Client displays
- For any update of page:
  - new request sent from client to server
  - server has to respond with complete page, HTML, styling etc
  - client has to render the page again from scratch
- Potential issues
  - Server load: lots of redundant data to be sent each time
  - server-rendering -> more work
  - slow updates: load full page, re-render

So, the original web, as designed in the late 80s, early 90s, had this notion that, it is just a client sends a request, the server responds, and the client displays whatever came back from the server. Now, anytime you want to update what is displayed on the screen, a new request has to be sent from the client to the server, the server has to respond with the complete page, which is basically at that point was HTML, even styling really did not exist at that point, you pretty much had to give everything, you just had the HTML tags.

And it was, in fact up to the browser, how it would choose to display various things, the whole idea of different fonts and all did not really exist at that time. So, the client has to render the page, again, from scratch. So, it once again, goes through parses the entire HTML file, and displays it. So, what are potential issues or here, there is some kind of server load in the sense that lots of redundant data is being sent each time. There could have been like, for example, it is quite likely that many pages might have the same header, footer, copy information, navigation bars, and so on. And all of that now needs to be sent again, from scratch each time. But, that is not really server load, it is more of network load, because after all, the server is just pretty much picking up static files and sending.

But on the other hand, when server rendering did come into the picture with CGI, it meant that the servers needed to do more work, they actually had to, put everything together, assemble it, and send it out, which effectively meant there was more work that needed to be done. And the end result, as far as the UX, the user experience is concerned, is that it also made pages slower to render, because the whole page has to be redisplayed from scratch for every request.

(Refer Slide Time: 02:35)



Asynchronous Updates

- Update only part of the page
  - Load extra data in the background after the main page has been loaded and rendered
- Quick response on main page: better user experience
- Request for update can ask for just minimal data to refresh part of a page
  - Example: show user a form to select animal
  - request data about animal alone from server - no need for HTML or other styling
  - refresh only one <div> in the page with text about the animal
- Originally seen as AJAX, now many variants

Core idea: refresh part of the document based on asynchronous (background) queries to server

So, then came the idea of an asynchronous update. At some point, people said, look, why not update only part of the page, load extra data in the background, after the main page has been loaded, and rendered. Which means that the first, the main page that you are going to see comes up pretty quickly, because it is a small amount of data. And you overall have a nice user experience. I mean, you click on a link loads very fast, I mean, we generally feel happy when we are browsing, and we come to a web page that loads fast and displays quickly, at least to start with.

So, let us say that, you had a form where you were asked to select a kind of animal, or a page, where essentially, it gives you a form that allowed you to select a kind of animal. Now, you have two options. One is that each of those selections actually leads to a different link, which loads the complete HTML and displays it on the page.

What would happen when you clicked on a link, it would basically the whole page would go, it would be very clear that you are going to a new web page, because the entire screen would get refreshed. And it would start re displaying the entire thing, including the navigation bars and everything else that you have from the beginning.

Now, instead, what could happen is, let us say that I click on the button to look for a cat, it requests data about that animal alone from the server. And in fact, that data could be in some kind of structured form. So, what is the structured form that we have over there, it might, for example, say, what is the common name of the cat, what is the Latin name of the cat, where is it commonly found, each of those could be a different structured piece of information that comes back from the server.

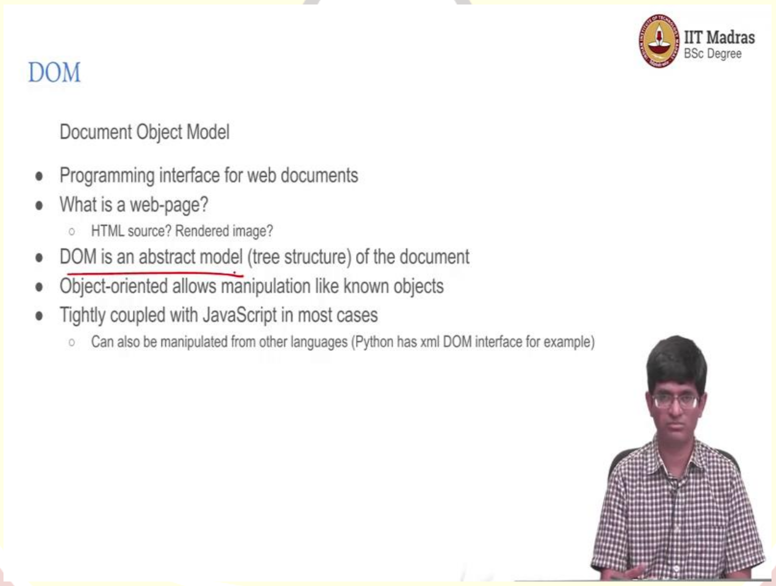
And all that you need to do is refresh one div, of course, divs did not exist at this point, they came up later, but still just one part of the page and put in the text corresponding to the animal. And even within that, you could have some part of it, which basically shows the name of the cat and other one which shows the Latin name of the cat in a different font, or in italics, something else which sort of gives you information about where it is commonly found.

So, all of that could be structured easily, and was sort of the precursor to the whole notion of API's where, you could have something where the server just returns the extra information needed in order to update the page. So, this originally came up and was known for a long time as AJAX. I mean, the term AJAX sort of still exists, but is not so common these days. Essentially what it stands for us Asynchronous JavaScript and XML. And why XML because the XML took care of the structured data coming back from the server.

So, this whole business of having structured information coming back from the server in the form of XML data, which can be easily passed, so that the Latin name, the place where it is found, and so on are easily distinguished, meant that the server now sends back data in that kind of format, it does not send a whole HTML page, it does not redo the styling. And as far as the client is concerned, it has most of the page already rendered, it just needs to replace one small part of it.

So, the core idea here was that you could refresh a part of the document based on an asynchronous query, which happens in the background. So, the asynchronous essentially means that it is happening in the background, it is not something which is in your critical path, it is not that you need to do that before you can start displaying the page, you display the main part of the page, perform this query, and then update the page based on the response.

(Refer Slide Time: 06:24)



DOM

Document Object Model

- Programming interface for web documents
- What is a web-page?
  - HTML source? Rendered image?
- DOM is an abstract model (tree structure) of the document
- Object-oriented allows manipulation like known objects
- Tightly coupled with JavaScript in most cases
  - Can also be manipulated from other languages (Python has xml DOM interface for example)

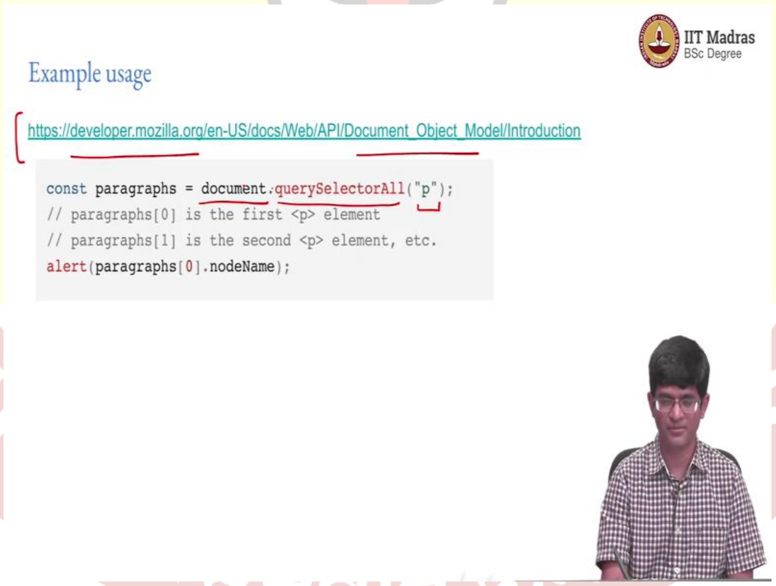
Now, all of this required a new concept in terms of how the document that is being displayed on the screen needs to be thought of, and that is what is called DOM or the Document Object Model. The DOM can be thought of as essentially a programming interface for web documents. And the question that it is trying to answer is, what exactly is a webpage, does it consist of just the HTML that you have sent, or does it consist of what it looks like after it is been rendered on the screen, or is there some kind of, an abstract tree like model, which is sitting somewhere inside the memory of your processor of your system, which is actually the webpage.

So, this DOM is, in fact, an abstract model. It sort of takes the HTML and parses it out, breaks it up into a tree, following the structure of the document. And now it says, since I have a tree structure, maybe I can use various kinds of algorithmic approaches are programming approaches, in order to manipulate that tree.

So, I could have something which basically tells me, this is the overall structure of the tree, and that these elements of the tree correspond to these parts of the screen. And if I modify one small portion of the tree in a certain way, I know exactly which parts of the screen need to get updated. And they were able to bring in an object-oriented approach to this, which allows manipulating the DOM, just like they are familiar with known object-oriented techniques.

For the most part, this is tightly coupled with JavaScript. And that does not mean that the DOM by itself is fundamentally limited to JavaScript. It is just that most browsers support JavaScript for manipulating the DOM. They do not support other languages directly. But there is, for example, a Python class within XML, within the XML module, which allows you to at least get the DOM information and then decide what you can do with it know, what kind of manipulation of the DOM is possible.

(Refer Slide Time: 08:33)



Example usage

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

```
const paragraphs = document.querySelectorAll("p");
// paragraphs[0] is the first <p> element
// paragraphs[1] is the second <p> element, etc.
alert(paragraphs[0].nodeName);
```

So as an example, and, these websites that I sometimes link over here, in particular, the Mozilla developer network, developer.mozilla.org is a very useful set of pages for anyone interested in developing for the web. It is, of course, done by the people who create the Firefox browser, which means that they are very knowledgeable about web technologies. And a lot of the information that is available over here also tells you about what kind of things browsers support. And they are also involved in the standardization process and so on.

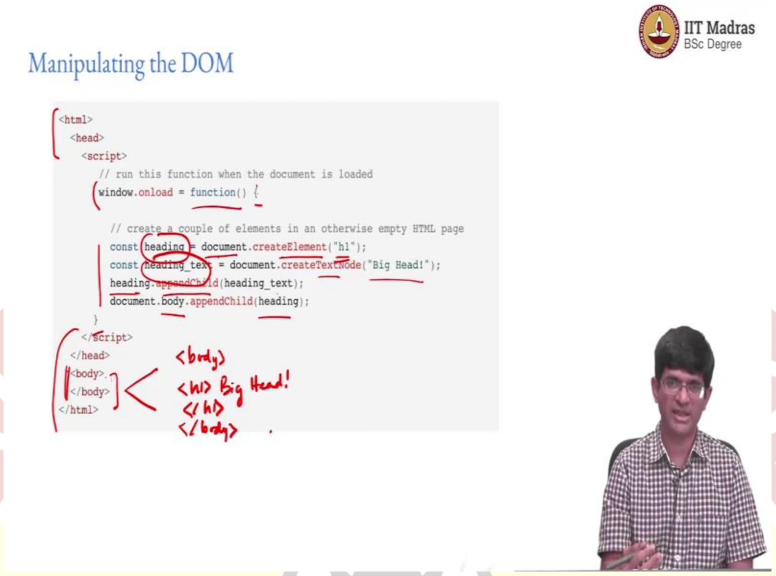
So, an example that they give on their web pages, you could have for something like this, where you basically in Java, this is for JavaScript code. If you are not familiar with JavaScript, do not worry about it. It is a reasonably easy to pick up language, in some ways similar to C, in some ways similar to Python. It is sort of a mix of both. But that is oversimplifying, obviously, I mean, when you read it, or when you start learning it, you understand what the differences are.

But the point is, it is not too difficult to pick up at least the basics. And all that I am going to be looking at over here is something that could easily be understood even in terms of something like C. And in this example, we are basically taking the document object and there is a function, which is part of the DOM manipulation API's inside JavaScript, which is query selector all and basically, we will pick out all elements inside the DOM that have the p type. And what is p, it is basically the paragraph type.

So therefore, and it assigns that to an array called paragraphs. So, paragraphs[0] will be the first p element, paragraphs[1] will be the second element, etc. and alert paragraphs[0].nodeName will basically pop up a small window showing the node name of the first part of the first paragraph in this case.

So, this is a very trivial example of what, over here, this document.(dot) this is essentially what the DOM is referring to. It is treating the document as an object and therefore I can do that object.(dot) and some function name, or a method name, which can be called on that object in order to get further information.

(Refer Slide Time: 11:04)



Manipulating the DOM

```
<html>
  <head>
    <script>
      // run this function when the document is loaded
      window.onload = function() {

        // create a couple of elements in an otherwise empty HTML page
        const heading = document.createElement("h1");
        const heading_text = document.createTextNode("Big Head!");
        heading.appendChild(heading_text);
        document.body.appendChild(heading);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

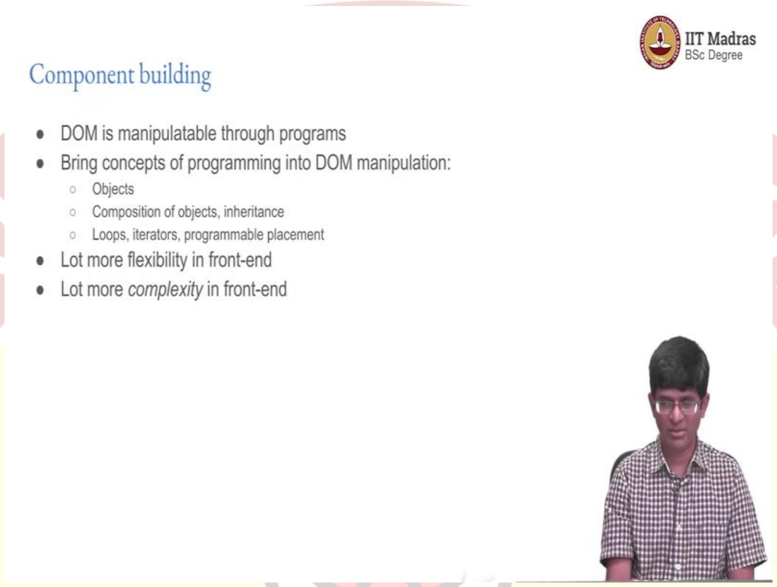
Now, I can get information. What is more interesting is when I start manipulating the DOM, I can also put things onto the screen. And this is an example of what could be done. So, all of this, this initial part HTML and the closing part are just required boilerplate around it. What is interesting is that when the window is loaded, you can tell it to run this particular function. Now, this syntax is something that is called an anonymous function in JavaScript.

What it says is, I am just going to create it as a function and put it within these curly braces, without giving it a name, so the function does not have a name, I am just declaring it as a function and assigning it directly to this window.onload. So, all that it means is when the window is loaded, all of these steps will get executed. What will actually happen, it will go to the document, it will create a new element of type h1, so in h1 heading, it will create another text node with the value big head.

And it will take whatever heading this page already had, or rather rewrite somewhere out here, whatever it had and, it will take this heading object and append heading text to that heading object. And it will take the body out here and append whatever it has just created inside the body, which means that after this has run, this will basically start end up looking like <body> </body>, but inside it will actually have <h1> </h1>.

And this big head will be there as the text inside the h1 tag. So, it has basically taken the HTML that you had out here, which did not have anything in the body and added something to it, that is manipulating the DOM. And that is where essentially, the power of AJAX or any kind of DOM manipulation than JavaScript does comes from. The fact that it can do that means that it can update pages dynamically.

(Refer Slide Time: 13:17)



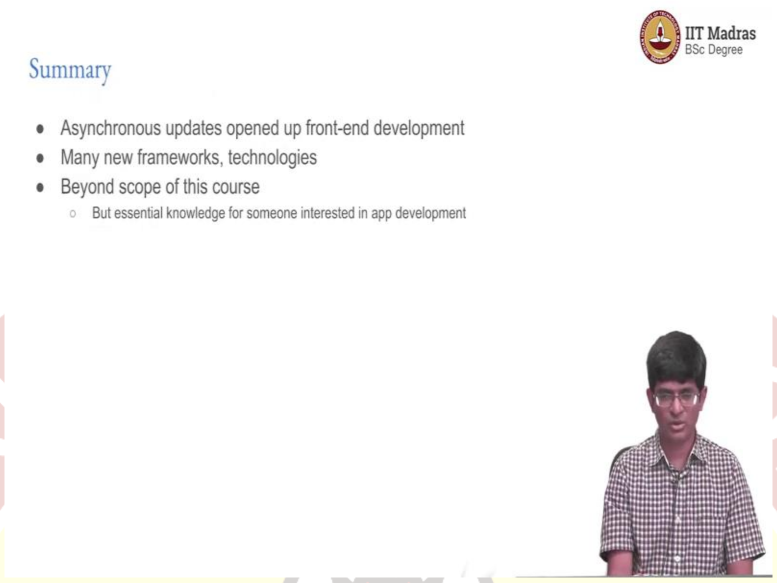
Component building

- DOM is manipulatable through programs
- Bring concepts of programming into DOM manipulation:
  - Objects
  - Composition of objects, inheritance
  - Loops, iterators, programmable placement
- Lot more flexibility in front-end
- Lot more *complexity* in front-end

Now, what that means is that, once you can manipulate the DOM, through programs, you can bring a lot of different concepts from programming into the manipulation, you can think of objects, you can compose objects, you can combine them together in different ways. You can have loops, iterators, all of this at runtime, in the client's browser.

Now, that gives you tremendous flexibility, in terms of what the frontend is capable of doing. On the other hand, it also adds a lot more complexity into what the frontend is doing, to what the frontend is capable of. And there is quite a good possibility that you complicate things too much. And, make the whatever you are trying to develop quite hard to understand. So, you always have to be careful of that, I mean, like the saying goes, with great power comes great responsibility, you need to be careful of how you use something like this.

(Refer Slide Time: 14:17)


IIT Madras
BSc Degree

Summary

- Asynchronous updates opened up front-end development
- Many new frameworks, technologies
- Beyond scope of this course
  - But essential knowledge for someone interested in app development

So, to summarize, asynchronous updates basically opened up front end development in a big way, they brought in a whole new ways of improving the user experience. And now a days, there are a whole lot of different frameworks and technologies that are built around this. It is no longer sort of goes by the name AJAX as such. But on the other hand, the core ideas developed from there, the fact that you could asynchronously get data and update whatever it is that's coming on the screen is what is being put to use.

Most of it unfortunately, it is beyond the scope of this course. It is essential knowledge for someone involved in app development. So, it is something that I will leave here, except that we will be looking at a little bit more about what is, what can be done and what are the implications on browsers and so on, but not really going into details on how to use this.