Intro to the DOM

Document Object Model

Before we DOM it up

- Rest/Spread Operators
- `Array.isArray` and Why We Need It?

Rest/Spread Operator(...)

The Rest/Spread operator (...) was introduced in ES6 and offers useful functionality.

Rest:
In functions, the Rest operator is used to gather any number of arguments that are passed after the defined parameters into an array. This allows for handling variable-length argument lists more easily.

Spread:
On the other hand, the Spread operator is used to spread or expand an array (or array-like object) into individual elements. Its simplest use case is to concatenate arrays, but it can be used in various other scenarios where array elements need to be separated or passed as individual arguments to a function.

Array.isArray()

- Yesterday, we learned that arrays in JavaScript are actually objects, as they are descendants of the Object prototype chain. This means they share some characteristics with objects. While typeof may not accurately determine if something is an array, JavaScript provides the Array.isArray(value) method specifically designed for this purpose. It allows us to check if a value is an array reliably.

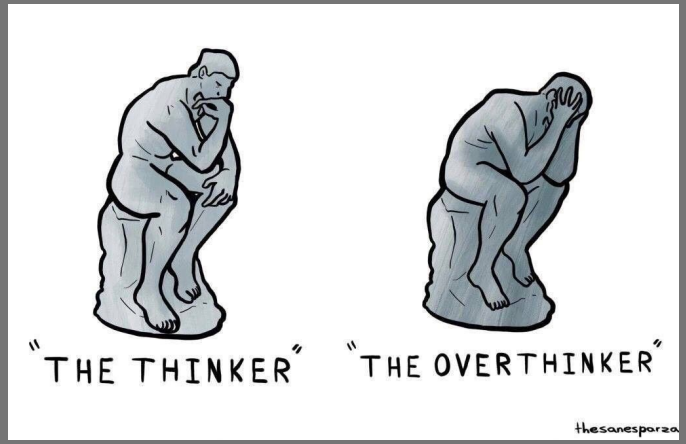
- If you're interested in learning more about the prototype chain and inheritance in JavaScript, you can refer to this Medium article: Master JavaScript Prototypes & Inheritance

You’re about to learn

- What is the DOM?
- Why do we care about the DOM?
- How do we manipulate the DOM?
  - Searching the DOM
  - How to traverse the DOM
  - How to change the DOM

The DOM

What is the DOM?



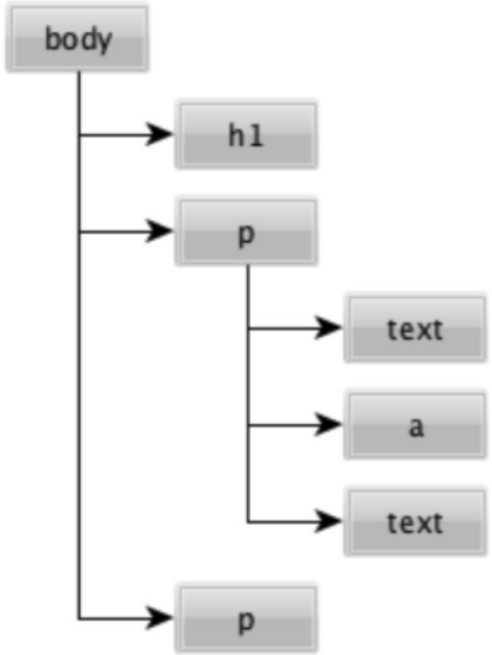
The DOM is...

The Document Object Model

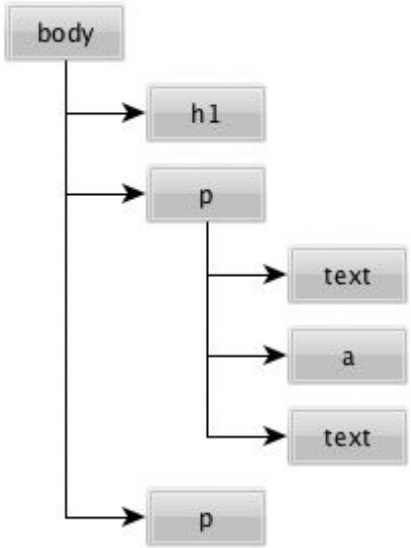
- It allows web pages to render
- Lets us respond to user events
- Change what we render based on user events

HTML vs DOM

```
<body>

  <h1>Hello</h1>

  <p>

    Check out my

    <a href="/page">Page!</a>

    It's the best page out there
  </p>



  <p>Come back soon!</p>
</body>
```



The DOM is a Tree



Main Idea:

There is a root node(body) that branches into other Nodes, known as children(h1, p)

Rules:

- Each Node can have 0 to many children
- Nodes can have 0 or 1 parent
- Nodes can have 0 to many siblings

Browser Dev Tools

Chrome

Right Click on Browser Window/Document and Inspect

Mac Quick Command: `Command+Option+J`

Windows/Linux/ChromeOS Quick Command: `Control+Shift+J`

FireFox

Tools > Web Developer > Toggle Tools

Mac Quick Command: `Command+Option+I`

Windows/Linux/ChromeOS Quick Command: `Control+Shift+I`

Why do we care about the DOM?

We care about the DOM because...

The DOM makes it possible to use Javascript to manipulate the document content and structure

Nodes have many attributes!

- Nodes in JavaScript are represented as JavaScript objects.
- They possess attributes that are defined as JavaScript properties.
- These attributes determine the appearance and behavior of the node in response to user interactions.

The *document* Object

- The document object serves as the global reference to the entry point of the Document Object Model (DOM).
- Provides methods that enable navigation and manipulation of the DOM.
- The document object plays a crucial role in connecting the JavaScript code we write with the DOM.

Searching the DOM

getElementbyID

Finds node with a certain ID attribute(IDs are unique)

`document.getElementbyId()` ⇐ we pass in the ID as the arg

getElementsByClassName

Finds nodes with a certain CLASS attribute

`document.getElementsByClassName()` ⇐ we pass in the class name as the arg

Searching the DOM (Continued)

getElementsByTagName

Finds nodes with a certain HTML tag

`document.getElementsByTagName()` ⇐ we pass in the html element e.g. div, h1, p

queryselector
querySelectorAll

Searching the DOM using CSS selectors

`document.querySelector()` ⇐ we pass in the CSS element e.g. #container, .container:first-child

Array-Like Objects?!?

- DOM selector methods return an HTMLCollection (or NodeList for querySelectorAll), which is an object representing a collection of nodes.

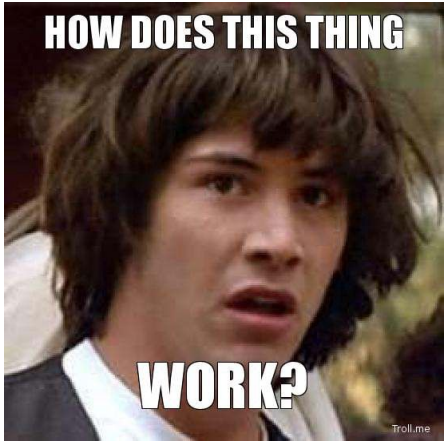
- This NodeList looks suspiciously like an array, but don't be fooled!

```
const divList = document.getElementsByTagName("div")
Array.isArray(divList) // false
```

- The NodeList is still zero-indexed, and values are accessible by index look-up like an array e.g **divList[0] gets you the first element**

- However, you won't have access to any array methods on the array prototype. This limits how you could programmatically operate over the NodeList

Array-Like Objects? A workaround



There are three ways to get around this:

1. `const divArr = [].prototype.slice.call(divList)`

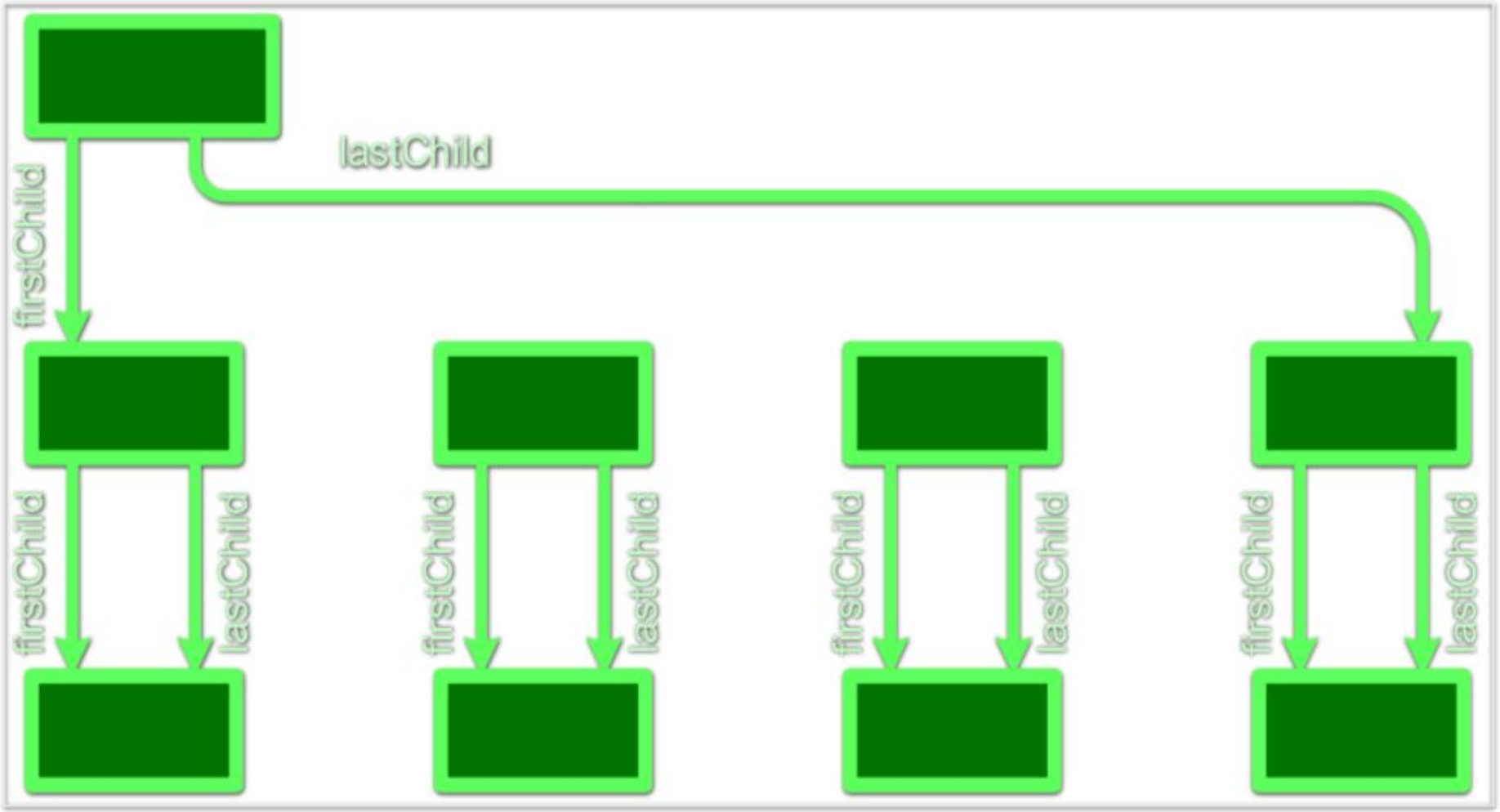
2. `const divArr = Array.from(divList)`

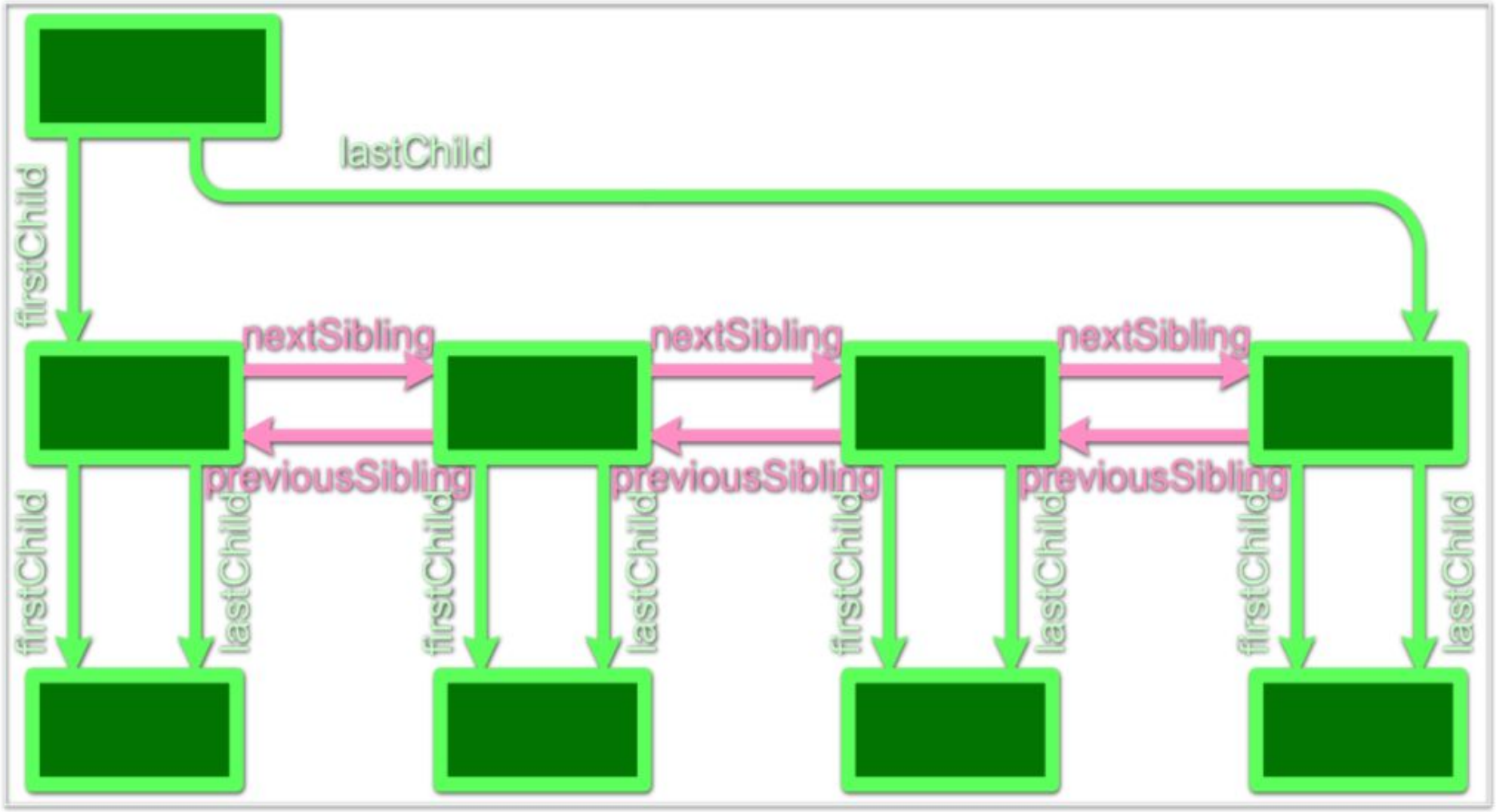
3. `const divArr = [...divList]`

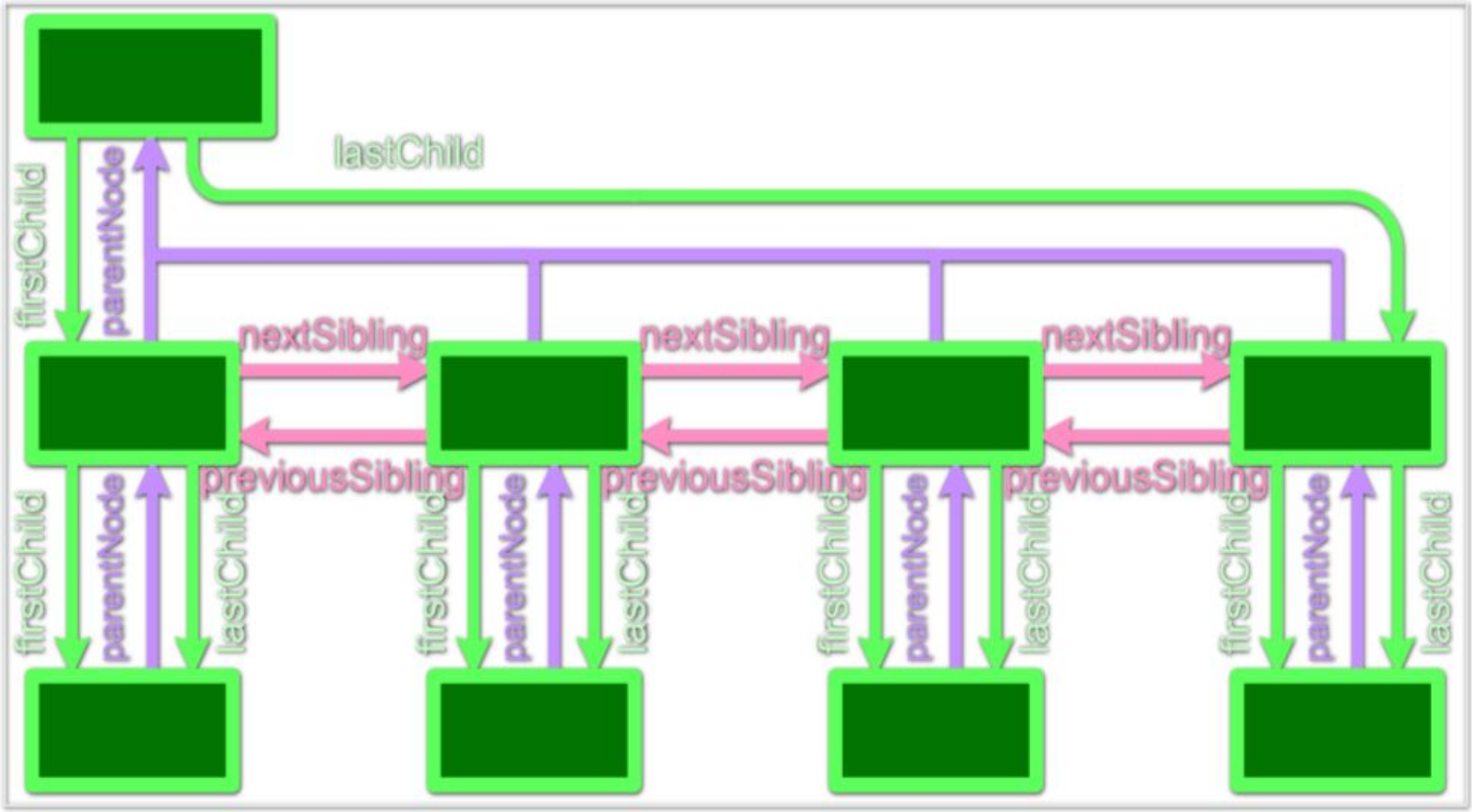
Traversing the DOM

As the DOM is a Tree Structure, it is relatively easy to navigate because:

- At any point in the DOM you are at a Node
- No matter where you go, you are still at a Node
  - Child… Node
  - Parent… Node
  - Sibling… Node
- All Nodes share similar DOM navigation methods


lastChild
firstChild
firstChild
lastChild
firstChild
lastChild
firstChild
lastChild
firstChild
lastChild


lastChild
firstChild
nextSibling
nextSibling
nextSibling
previousSibling
previousSibling
previousSibling
firstChild
lastChild
firstChild
lastChild
firstChild
lastChild
firstChild
lastChild


lastChild
firstChild
parentNode
nextSibling
nextSibling
nextSibling
previousSibling
previousSibling
previousSibling
firstChild
parentNode
lastChild
firstChild
parentNode
lastChild
firstChild
parentNode
lastChild
firstChild
parentNode
lastChild

Traversing the DOM

Access children Nodes

element.children
element.lastChild
element.firstChild

Access sibling Nodes

element.nextElementSibling
element.previousElementSibling

Access parent Node
If it exists!

element.parentElement

Changing the DOM: Style Attributes

```
document.getElementById(id).style.property = new style
```

Example:

```
document.getElementById("p2").style.color = "blue"
```

We are getting the p2 id and accessing its style.color and setting it to blue

What is happening here?

```
<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>
```

Changing the DOM: Removing Elements

Remove a child called oldNode

```
node.removeChild(oldNode)
```

Quick Hack if you don’t know the parent

```
oldNode.parentNode.removeChild(oldNode)
```

JS Event Handling

What is an event?

An event is...

A Javascript event is a callback that gets fired when something happens related to the DOM on your website

For Example:

Clicking an element like a button
Hovering over a link or button

We can attach event handlers to an element so that when a specific user event happens, the intended specific callback gets “fired”

Listening for Events - Native JS

```
document.getElementById(“myId”).addEventListener(“click”,
function(event){alert(‘I am clicked’)}
```

```
addEventListener(type, cb)
type : the event
cb: the anonymous function that fires when type occurs
```

The addEventListener attached to an event handler(an anonymous function to execute) when the type(“click”) occurs by the user

There are many events to listen to like `hover, keyup, keydown, mouseover, scroll`

The HTML <form> Element

The login, signup, and address forms you see online all share a common tag <form>

Inside of <form> are several elements that make up forms:

- Text input boxes
- Dropdown
- Radio buttons
- Checkboxes
- etc

<form> example

```
<form action="/process" method="POST">
  <label for="username">Username</label>
  <input type="text" name="username" id="username">
  <label for="password">Password</label>
  <input type="password" name="password">
  <input type="submit" value="Submit">
</form>
```

Don’t worry about action and method for now- also don’t worry about submitting your form just yet.

How do get inputs from a form?

You can see what is inside a form element fairly easily, using the `.value` attribute

```
<!-- Sample form input element -->
<input type="text" name="username" id="username">

document.getElementById('username').value
// returns the value of the field
```

Get the title of the form

Imagine a `<form>` with an `<h1>` tag above it that has the form title. We can use the attribute `.innerText` to retrieve the title inside the `<h1>` tag, or even change it.

Get the title of the form

```
<h1 id="title">Enter your information</h1>


heading = document.getElementById('title')
heading.innerText
>> "Enter your information"

// set name to Zach
var name = "Zach"

// Changes the text in the DOM
heading.innerText = "Enter " + name + "'s Information"

// inside of <h1> to say this instead
heading.innerText
>> "Enter Zach's Information"
```

Changing the content of a `<div>`

Let’s now say that our <h1> lives inside a . Using the .innerHTML attribute, we can change the innerHTML of the entirely.

Changing the content of a

Before:

```
<div id="main-section">
  <h1>Hello World</h1>
</div>
```

JS:

```
document.getElementById('main-section').innerHTML = "<h3>Hello World Smaller</h3>"
```

After:

```
<div class="main-section">
  <h3>Hello World Smaller</h3>
</div>
```

Code Demo