Rex Jones II

Selenium How To Find An Element In Shadow DOM

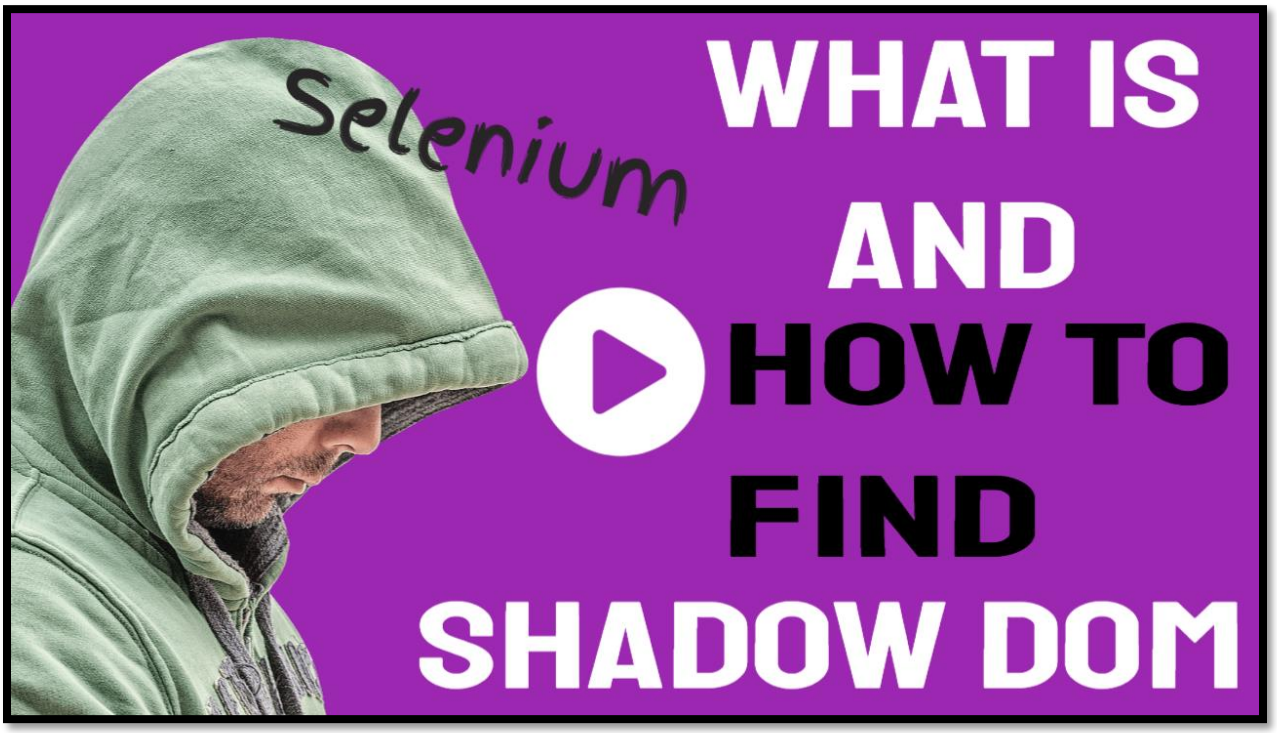


Table of Contents

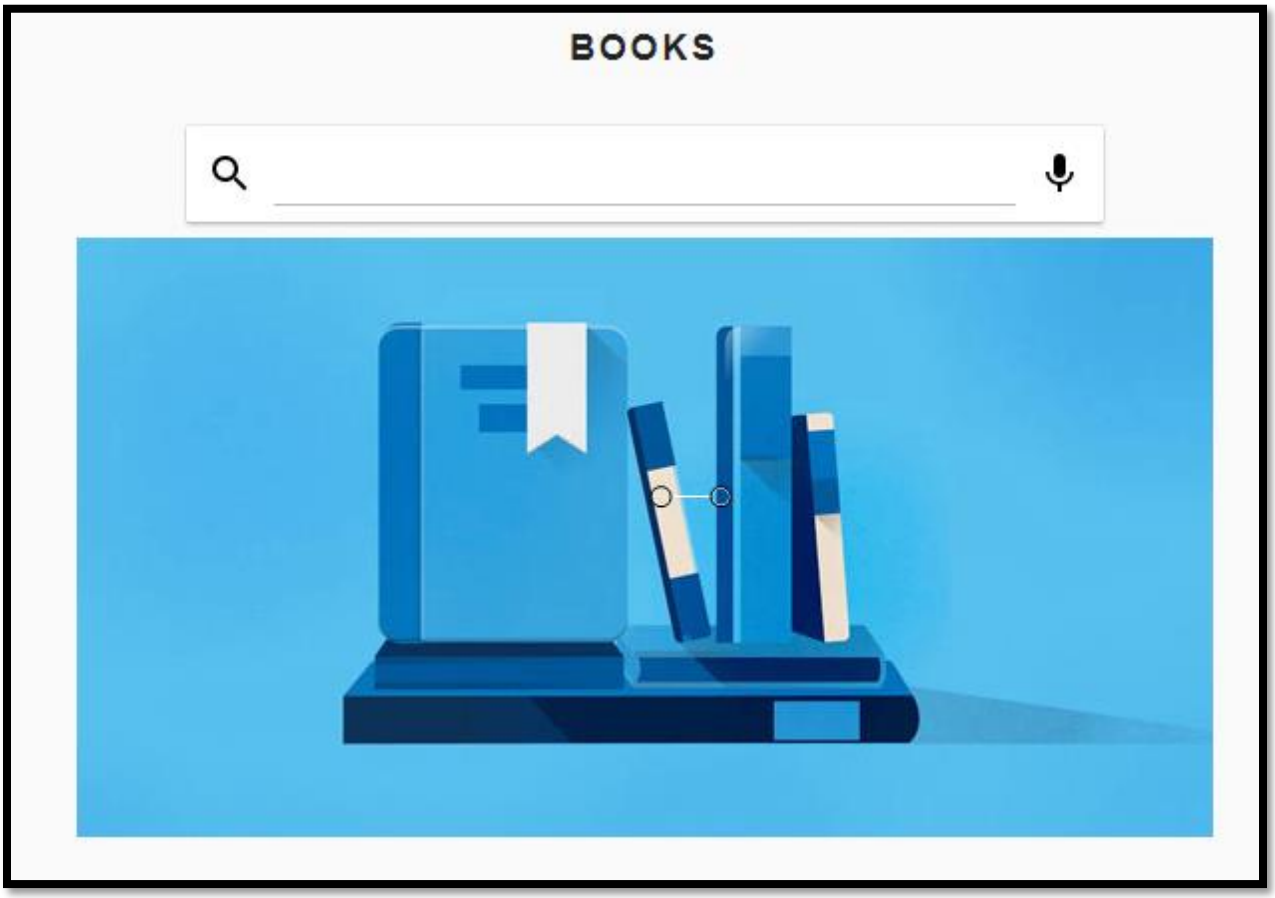
Shadow DOM Playlist

https://www.youtube.com/watch?v=sYkZhyoBMXM&list=PLfp-cJ6BH8u_MvNXdpHtc-0Q1uMCWb9dM&index=1

Introduction

In this session, I am going to introduce Shadow DOM. We can think of a Shadow DOM like a DOM within another DOM. That's why it challenging to locate an element inside of a Shadow DOM. DOM stands for Document Object Model which represents the HTML Nodes in a tree format. The purpose is to define properties, define a structure, and define contents of a web page. When it comes to a Shadow DOM, it allows a developer to hide the DOM by attaching it to a tree within the original DOM.

Let me show you how a developer can hide the DOM. On this popular Books page, we can inspect the search field.

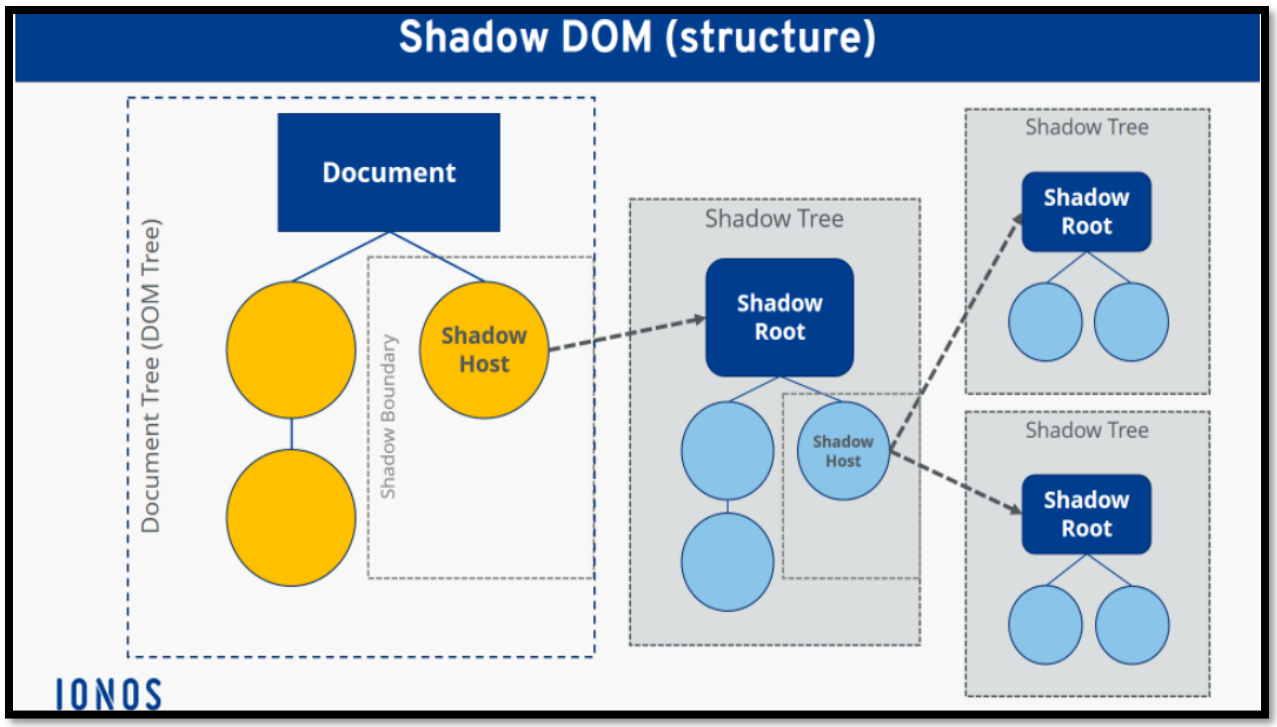


Notice above this highlighted section, we see shadow-root 4 times. However, this bread crumb gives us a clue that the search field element is in a Shadow DOM but we only need 1 of these shadow roots. Do you see hashtag #shadow-root? That is the only shadow-root with an expanded node. The other #shadow-roots are not relevant for locating the search field.

```
<!DOCTYPE html>
<html lang="en">
▶ <head>…</head>
▼ <body data-new-gr-c-s-check-loaded="14.991.0" data-gr-ext-installed>
  ▼ <book-app apptitle="BOOKS">
    ▼ #shadow-root (open)
      ▶ <style>…</style>
        <!-- Header -->
      ▼ <app-header condenses reveals effects="waterfall" style="transform: translate3d(0px, 0px, 0px);">
        ▶ #shadow-root (open)
          ::before
        ▶ <app-toolbar class="toolbar-top">…</app-toolbar>
        ▼ <app-toolbar class="toolbar-bottom" sticky style="transform: translate3d(0px, 0px, 0px);">
          ▶ #shadow-root (open)
          ▼ <book-input-decorator top>
            ▶ #shadow-root (open)
              <input slot="input" id="input" aria-label="Search Books" autofocus type="search"> == $0
            ▶ <speech-mic slot="button" continuous interimresults>…</speech-mic>
            </book-input-decorator>
          ▶ <h4 class="subtitle" hidden>…</h4>
          </app-toolbar>
        </app-header>
```

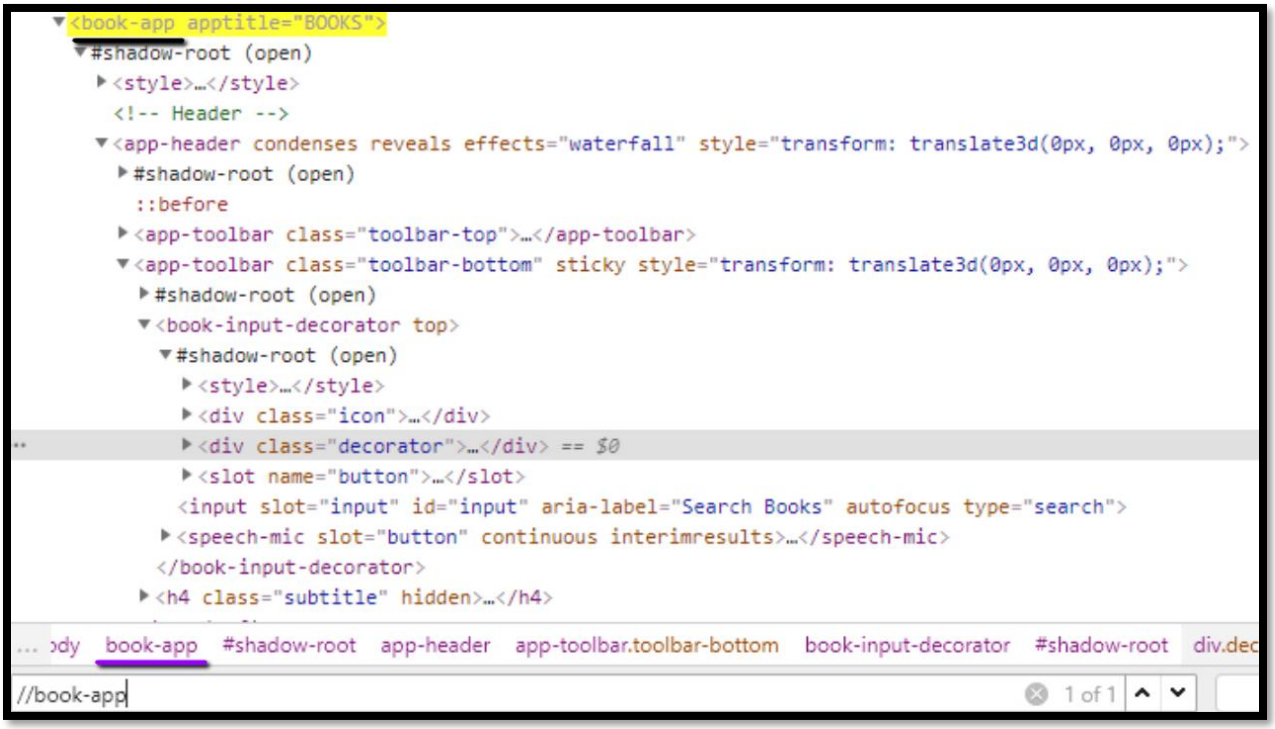
html body book-app #shadow-root app-header app-toolbar.toolbar-bottom book-input-decorator input#input

I found this diagram on the internet and it shows the Shadow DOM structure.



We have 1 original DOM and 3 Shadow DOM's. The original DOM has an orange Shadow Host. A Shadow Host is an element with a Shadow Tree. The Shadow Tree is a DOM Tree but that Shadow Tree always begin with a Shadow Root which is similar to the original DOM always begin with html. It has its own DOM tree with elements and a possible Shadow Host. Do you see how the 2nd box which is the 1st Shadow DOM has an extension from the Shadow Host? It has 2 more Shadow Roots. We are not able to locate no elements in the gray boxes because the developers chose to hide those elements within the Shadow DOM. However, we can locate the Shadow Host in the Original DOM.

For example, if I go back to our AUT, we see book-app is the Shadow Host in the Original DOM. Therefore, it's no problem finding this Shadow Host. Search by writing //book-app and we see the element highlighted yellow in the DOM.



Whenever we see Shadow Root that means it is the start of a new Shadow DOM. Watch what happens when I search for app-header by writing //app-header. It's no good; It's no good because the Shadow DOM hid the app-header element. It also hides the app-toolbar element and the book-input-decorator element. I can show you that also. //app-toolbar – did not find it and also look for //book-input-decorator and it did not find it.

From this diagram, we see how an element can be in 1 Shadow DOM or within a nested Shadow DOM. A nested Shadow DOM is when a Shadow DOM is located in another Shadow DOM. Something like the last 2 boxes. There are different ways to find an element inside a Shadow DOM. We can find an element without using the Selenium findElement Method. Also, locate an element when using the Selenium

findElement Method. I'm going to show you both ways and demo how to locate an element within a nested Shadow DOM.

Shadow DOM - How To Locate An Element Without Using Find Element

In this session, we are going to start by locating an element in the Shadow DOM without using Selenium findElement method. I am going to use the same book application from our Introduction and find this search field.

Inspect. We are not able to locate this search field from the Elements tab because the search field is in a Shadow DOM. Do you see how the id attribute has a value of input? If this were the original DOM, we could use xpath or cssSelector to find this element. For cssSelector, start with the tag name input then id # and the value input. Did not find it. Remove the input tag name and still no success. For xpath, we write //input[@id='input'] and once again we are not able to find the search field.

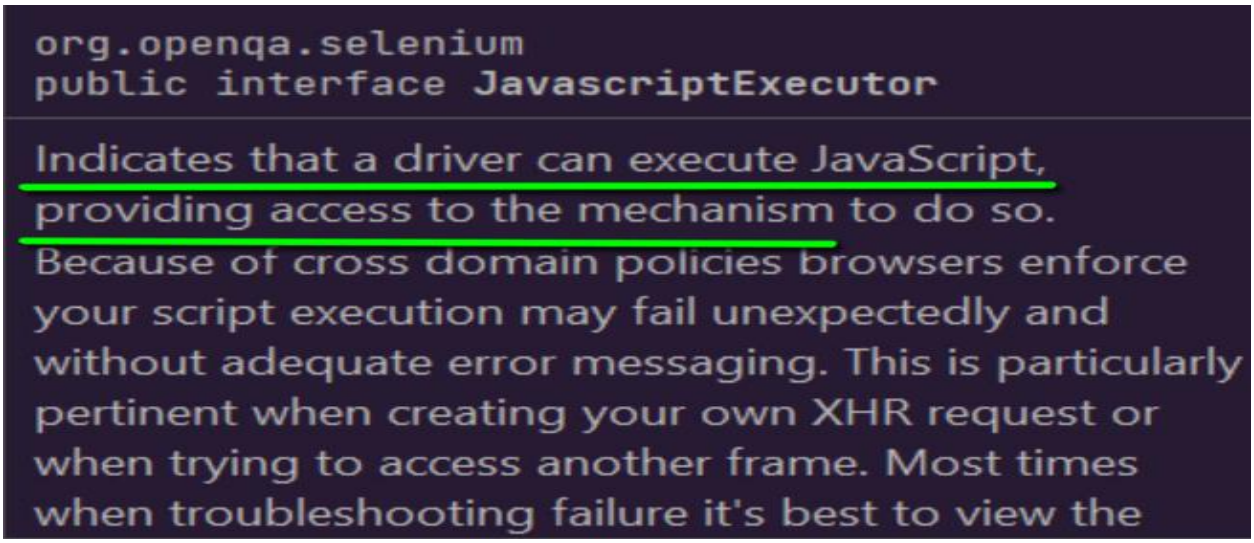
In our diagram, the search field would be an element in the 1st Shadow DOM. Now, let us write our code to find the search field. The setUp method is pre-written for Chrome and it loads the book application. The tearDown method is commented out to quit the driver.

```
public class ShadowDOM {

  WebDriver driver;

  @BeforeMethod
  public void setUp () {
    WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();
    driver.manage().window().maximize();
    driver.get("https://books-pwakit.appspot.com/");
  }

  @AfterMethod
  public void tearDown (){ //driver.quit();
  }
```

Our method is @Test public void findShadowDOMWithoutSeleniumFindElement () { } We have 3 steps to find an element in the Shadow DOM. 1st we // Provide Access So The Driver Can Execute JavaScript 2nd is to // Execute JavaScript To Return A WebElement 3rd we // Perform An Action On The WebElement

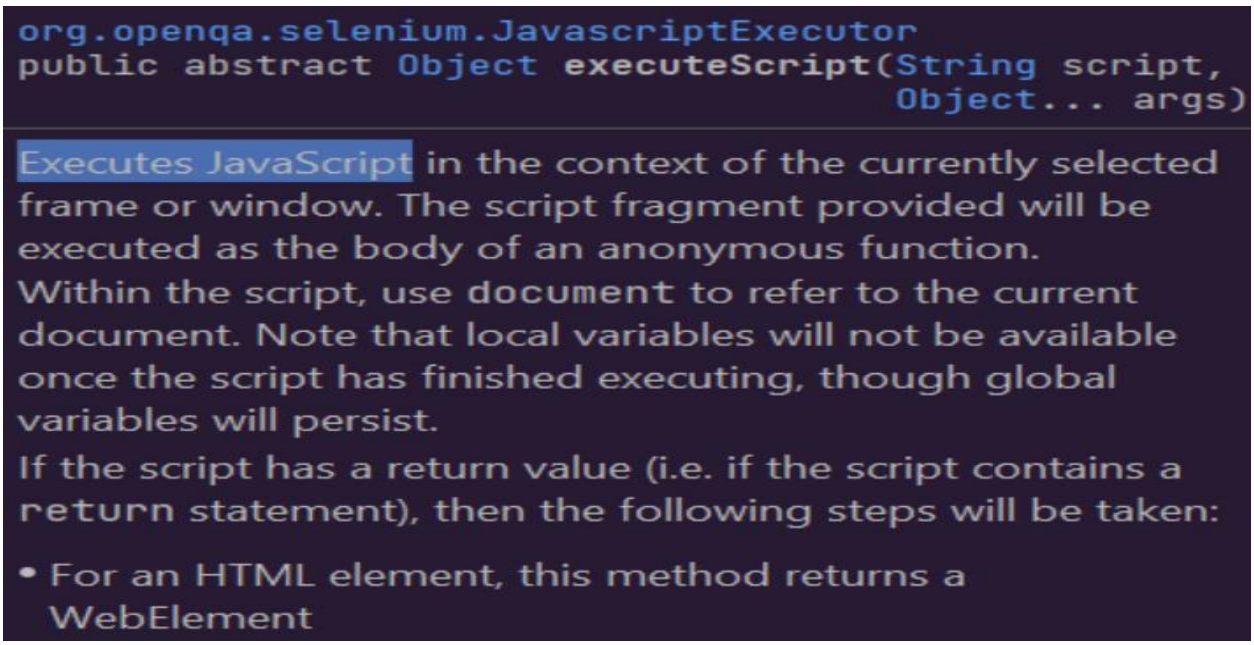
```
@Test
public void findShadowDOMWithoutSeleniumFindElement () {
  // Provide Access So The Driver Can Execute JavaScript

  // Execute JavaScript To Return A WebElement

  // Perform An Action On The WebElement
}
```

JavaScriptExecutor is an interface with jsExecutor; as the reference; JavaScriptExecutor indicates that a driver can execute JavaScript, providing access to the mechanism.



So we convert to = (JavaScriptExecutor) from the driver; At this point, the driver has access to execute JavaScript.

Now, that we have access the jsExecutor. will execute the Script(“”);The executeScript method executes JavaScript. I want you to notice 2 things in this intellisense. #1 Within the script, we use document to refer to the current document and #2 If the script has a return value, then the following steps will be taken. Our test will return an HTML element, so this method returns a WebElement.



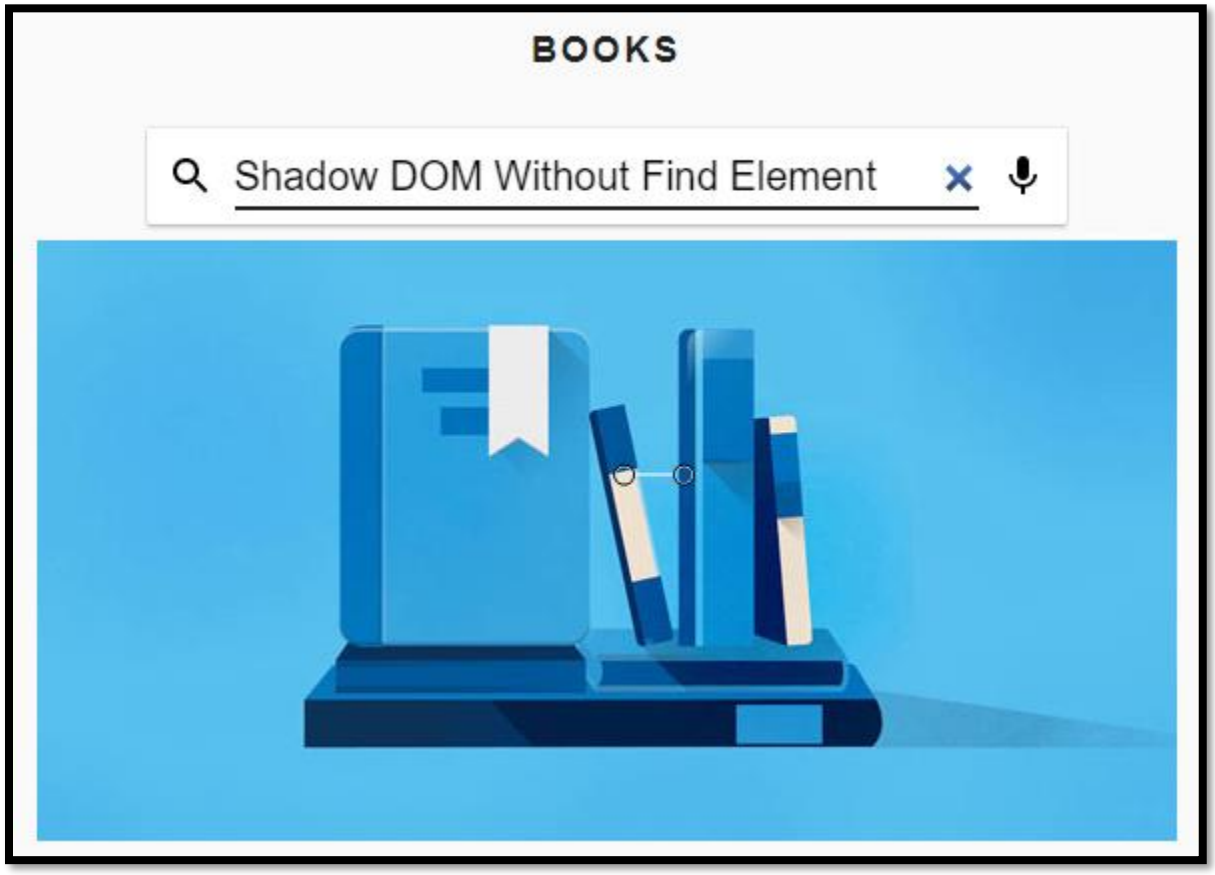
To sum up the information from intellisense, our executeScript method will “return a WebElement and use document”. Since the script returns a WebElement, we cast (WebElement) then assign it to WebElement with a name like bookSearchField =. To complete this script, we need the path from JavaScript.

Go back to our AUT. I’m going to show you 2 more ways for finding this path but for now let’s just keep it simple and we right click the element, select Copy, Copy JS path. That’s it. Go back to our code and Paste the script. This is the path for our bookSearchField and the script returns a WebElement, document represents our web page, querySelector allows us to find the book-app element using cssSelector, shadowRoot is a property representing the shadow root element. Finally, we have the querySelector to find the search field element using cssSelector hashtag #input.

Last step is to perform an action on the WebElement by writing bookSearchField.sendKeys(“Shadow DOM Without Find Element”).

```
@Test
public void findShadowDOMWithoutSeleniumFindElement () {
  // Provide Access So The Driver Can Execute JavaScript
  JavascriptExecutor jsExecutor = (JavascriptExecutor) driver;

  // Execute JavaScript To Return A WebElement
  WebElement bookSearchField = (WebElement) jsExecutor.executeScript(
          script: "return document.querySelector(\"body > book-app\")." +
                  "shadowRoot.querySelector(\"#input\")");

  // Perform An Action On The WebElement
  bookSearchField.sendKeys( ...keysToSend: "Shadow DOM Without Find Element");
}
```

Let's Run. Bingo, in the search field we see Shadow DOM Without Using Find Element.



That's it. Thanks for watch and I'll see you in the next session.

Shadow DOM – How To Locate An Element With Find Element

Locating an element using Selenium's findElement method requires us to walkthrough the DOM step-by-step. For an illustration using our diagram, we are going to find the Shadow Host then locate each element in the Shadow Tree. From the previous session, our setup method controlled Chrome and loaded the book application.

```
@BeforeMethod
public void setUp () {
  WebDriverManager.chromedriver().setup();
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://books-pwakit.appspot.com/");
}
```

In this session, we will use the same setup and the Test Script will be @Test public void findShadowDOMWithSeleniumFindElement () {}. We are going to group all of the statements into 3 statements starting with // #1 Find Shadow Host // #2 Execute JavaScript To Return The Shadow Root // #3 Find The WebElement Then Perform An Action On The WebElement

```
@Test
public void findShadowDOMWithSeleniumFindElement () {
  // #1 Find Shadow Host

  // #2 Execute JavaScript To Return The Shadow Root

  // #3 Find The WebElement Then Perform An Action On The WebElement
}
```

The Shadow Host is a WebElement so we write WebElement shadowHost; Now we have to find the element. Recall from the diagram, the shadow host comes before the Shadow Tree and a Shadow Tree is a tree in the Shadow DOM. All Shadow Trees begin with a shadow-root. Go to our AUT. Inspect the search field. We see book-app comes before shadow-root.

```
<!DOCTYPE html>
<html lang="en">
▶ <head>…</head>
▼ <body data-new-gr-c-s-check-loaded="14.991.0"
  ▼ <book-app apptitle="BOOKS">
    ▼ #shadow-root (open)
      ▶ <style>…</style>
        <!-- Header -->
```

Therefore, our code will be driver.findElement(By.tagName("book-app"));

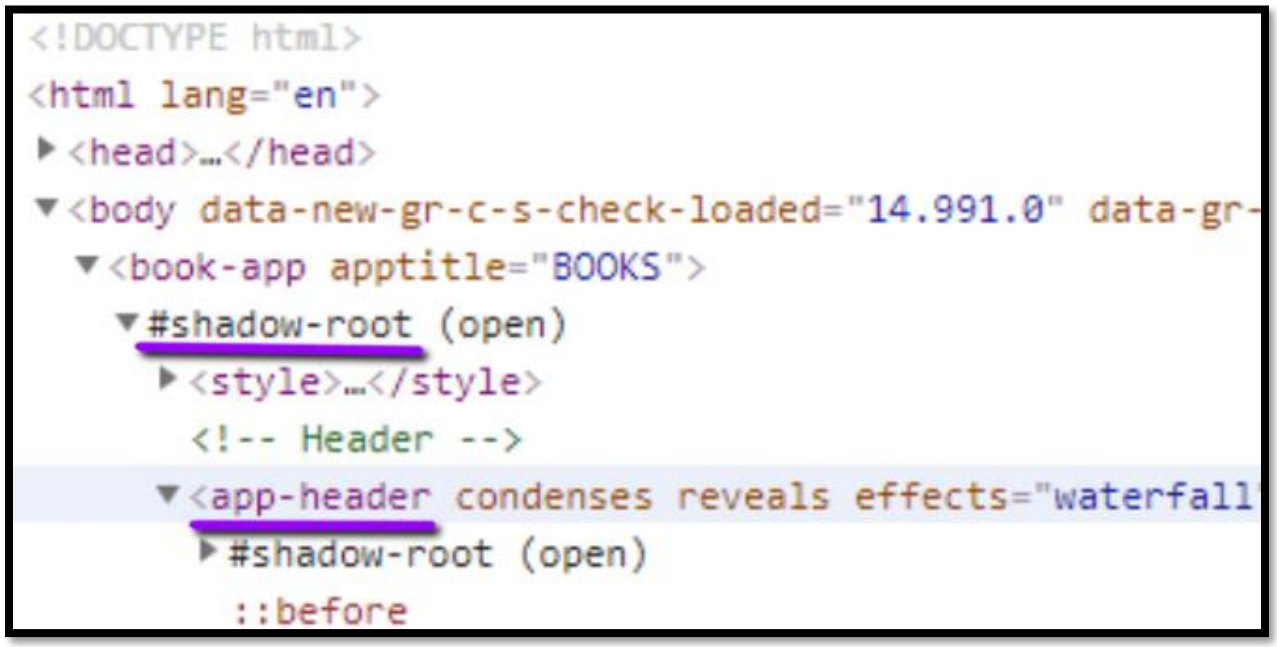
```
@Test
public void findShadowDOMWithSeleniumFindElement () {
  // #1 Find Shadow Host
  WebElement shadowHost = driver.findElement(By.tagName("book-app"));
```

We have 2 statements for JavaScript to return the Shadow Root. The first statement is to cast (JavascriptExecutor) from the driver then assign it to JavascriptExecutor with any reference name like jsExecutor =. The next statement will cast (WebElement) and use jsExecutor. to execute Java Script(). We execute JavaScript with 2 parameters. ("return arguments[0].shadowRoot", and shadowHost).

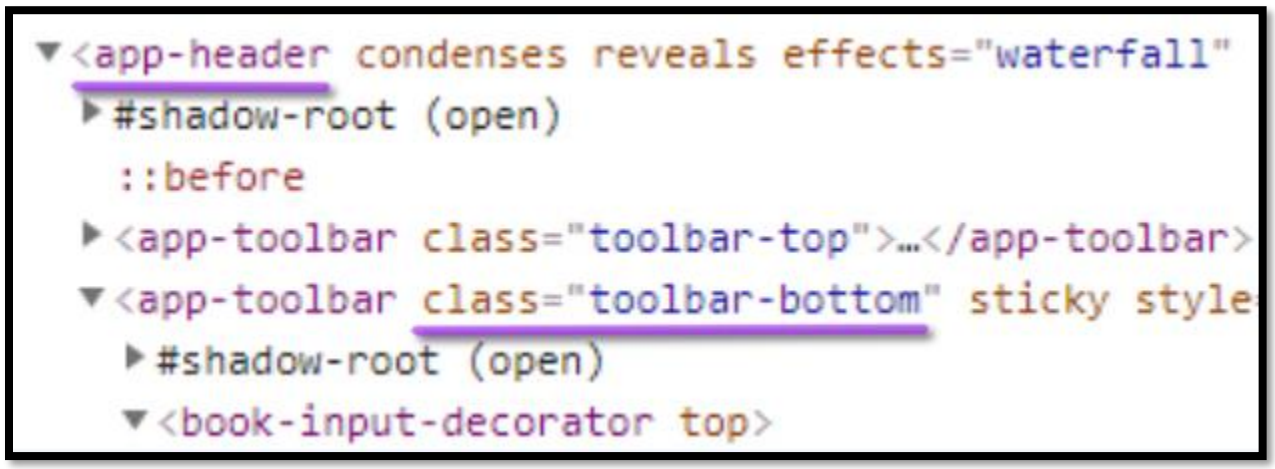
The first parameter returns the 1st argument representing the Shadow Root element. Notice the 2nd parameter is shadowHost. We need the shadowHost to access the WebElement for shadowRoot =. The shadowRoot is located inside of the Shadow Host.

```
@Test
public void findShadowDOMWithSeleniumFindElement () {
  // #1 Find Shadow Host
  WebElement shadowHost = driver.findElement(By.tagName("book-app"));

  // #2 Execute JavaScript To Return The Shadow Root
  JavascriptExecutor jsExecutor = (JavascriptExecutor) driver;
  WebElement shadowRoot = (WebElement) jsExecutor.executeScript(
          script: "return arguments[0].shadowRoot", shadowHost);
```

Now we can use the shadow root for the WebElement app_header; Go back to our AUT and we see app-header is below shadow-root.

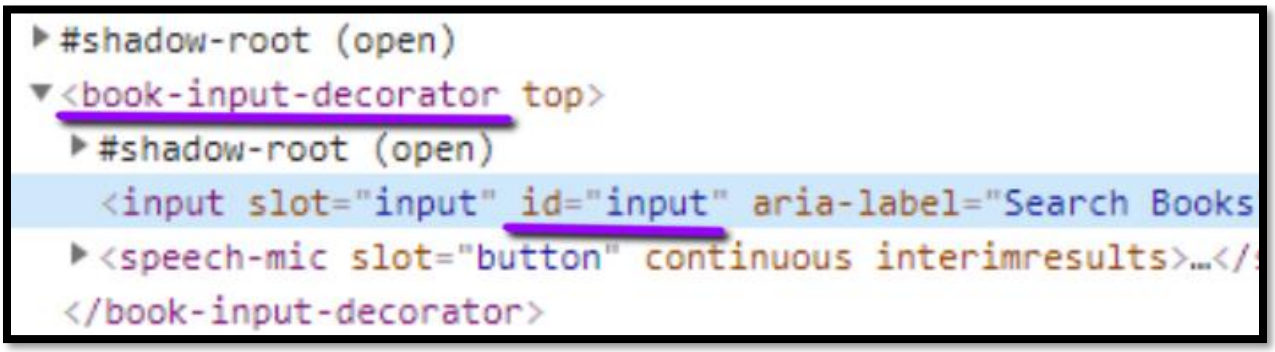


For our code, each WebElement will build on top of the previous WebElements. We are going to use shadow-root and the app-header tag name. = shadowRoot. then find the element (By.tagName("app-header")); The next element is app-toolbar but there are 2 tags with the same name. So we must identify the element using the class name which is toolbar-bottom. Just like the last statement. We build on the current element using the previous element. So we use app-header and toolbar-bottom.



WebElement app_toolbar = previous WebElement app_header.findElement(By.) At this point, we can find the element by Selenium locators className or cssSelector. className would be ("toolbar-

bottom") but let's use cssSelector(".toolbar-bottom")); The next tag is book-input-decortator then the book search field which has an id value of input.



We are going to build on top of app-toolbar then find book-input-decorator followed by the search field. So the WebElement will be WebElement book_input_decorator = app_toolbar.findElement(By.). If we wanted to, we can combine the book-input-decorator element with the search field using cssSelector(""). The value would be both tag names ("book-input-decorator > input"). However, let's break it down and use only the tagName("book-input-decorator"). Now, we are at the last WebElement which is the WebElement for searchField = book_input_decorator.findElement(By.id("input")). Now that we have found our WebElement, let's perform an action on the WebElement by writing searchField.sendKeys("Shadow DOM With Find Element");

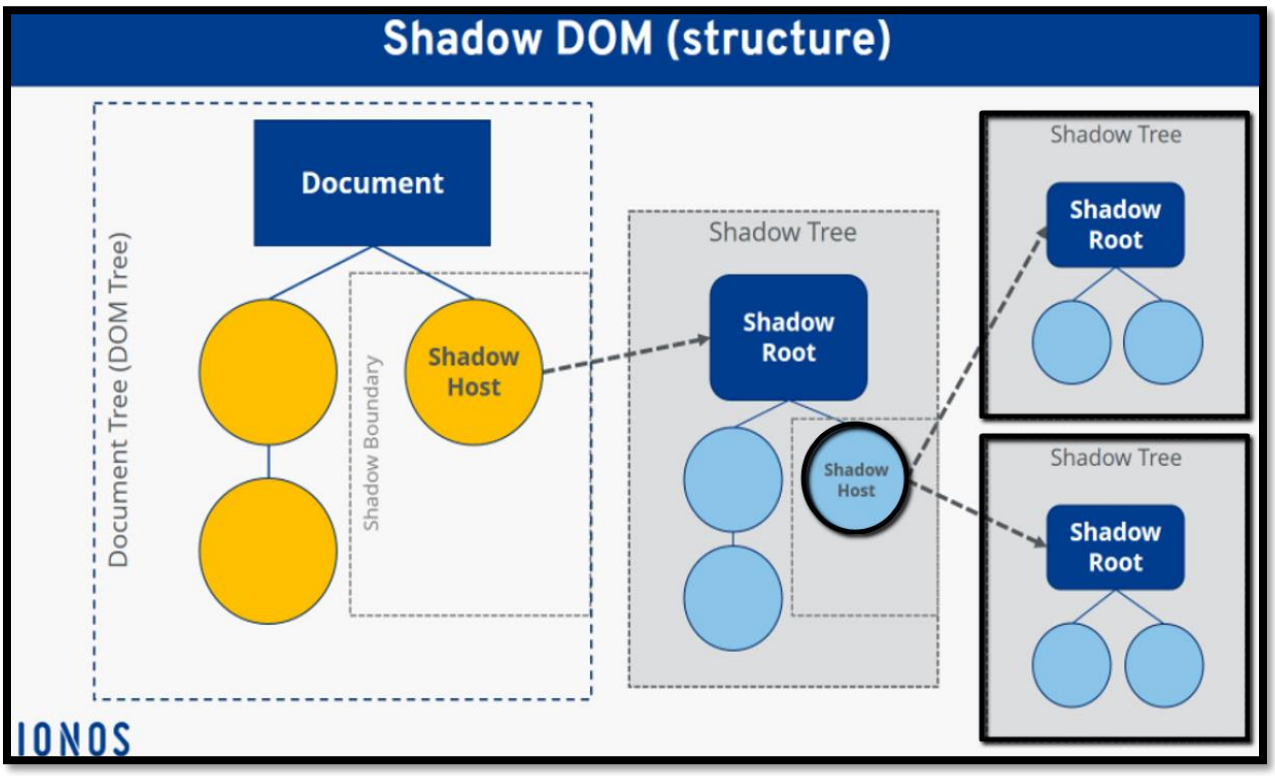
```
// #3 Find The WebElement Then Perform An Action On The WebElement
WebElement app_header = shadowRoot.findElement(By.tagName("app-header"));
WebElement app_toolbar
        = app_header.findElement(By.cssSelector(".toolbar-bottom"));
WebElement book_input_decorator
        = app_toolbar.findElement(By.tagName("book-input-decorator"));
WebElement searchField = book_input_decorator.findElement(By.id("input"));
searchField.sendKeys( ...keysToSend: "Shadow DOM With Find Element");
```

Let's run. We see the search field shows Shadow DOM With Find Element. Thanks for watching and I'll see you in the next section for Nested Shadow DOM's.

Nested Shadow DOM

In this session, we are going to find an element in a Nested Shadow DOM. The previous 2 sessions showed different ways of finding an element. We found an element using Selenium's findElement method and without the findElement method. I am going to locate the nested Shadow DOM element using a JavaScript expression.

A nested Shadow DOM is when the Shadow DOM is located inside of another Shadow DOM. For example, in this diagram we see the last 2 gray boxes are extended from the first gray box. The first gray box is a Shadow DOM and has a Shadow Host. It splits into more Shadow DOMs.



For our code, it is set up to use Chrome, maximize the window, and to load the application.

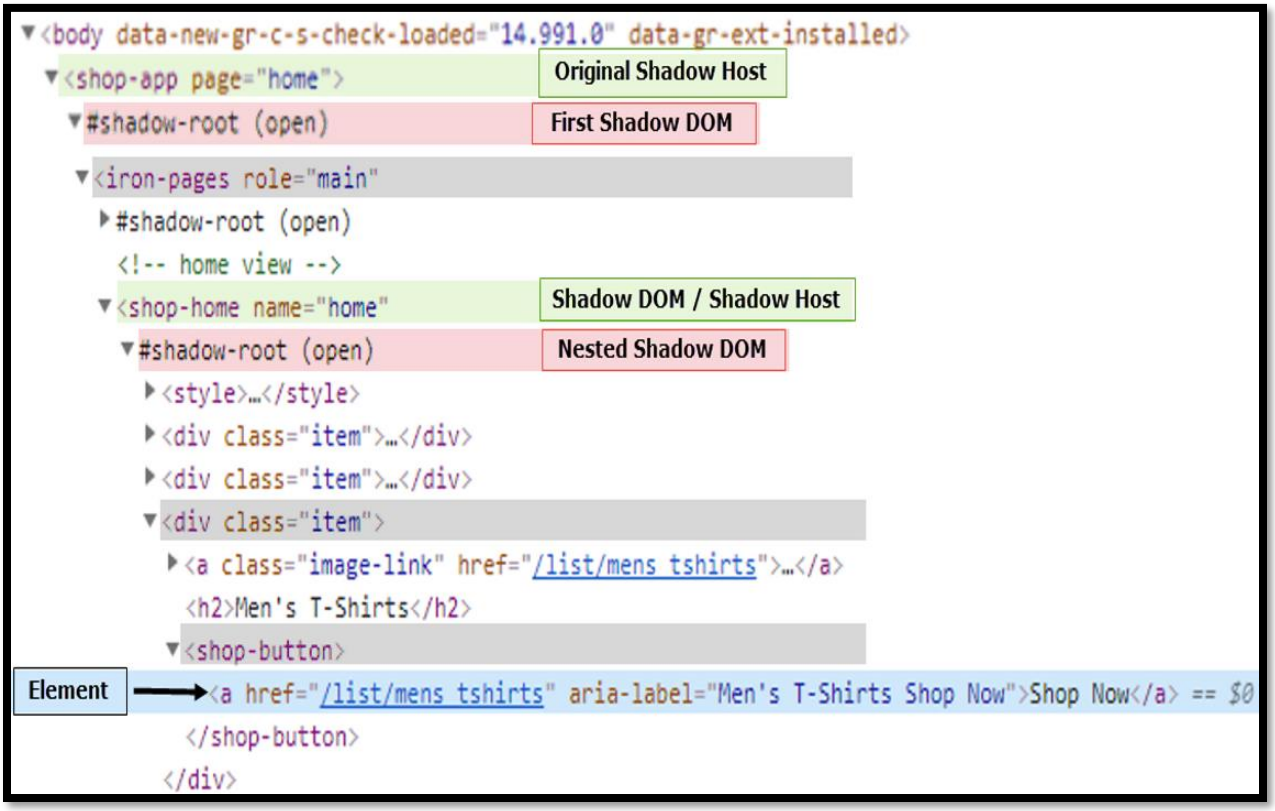
```
@BeforeClass
public void setUp () {
  WebDriverManager.chromedriver().setup();
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://shop.polymer-project.org/");
}
```

For our test, it starts @Test public void findElementInNestedShadowDOM () {}. There will be 2 main steps // Provide Access To JavaScript and // Find The WebElement So We Can Perform An Action

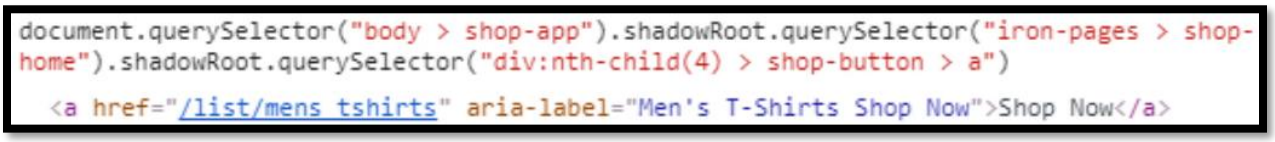
```
@Test
public void findElementInNestedShadowDOM () {
  // Provide Access To JavaScript
  JavascriptExecutor jsExecutor = (JavascriptExecutor) driver;

  // Find The WebElement So We Can Perform An Action

}
```

The (JavascriptExecutor) interface indicates a driver can execute JavaScript. Therefore we cast it then assign it to JavascriptExecutor with a reference jsExecutor =. Now, we must find the WebElement. Let's go to our AUT.

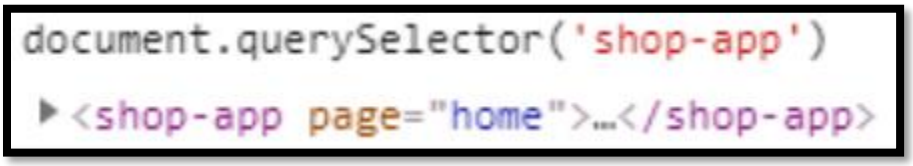
It's a Shop website for Men and Ladies. It has 4 Shop Now buttons and each Shop Now button is located inside of a nested Shadow DOM. We are going to click the Shop Now button for Men's T-Shirts. Inspect the Shop Now button. A fast indicator that the element is in a Shadow DOM is by looking at the bread crumb. We see shadow-root. However, it's a nested Shadow DOM because we see shadow-root 2 times. The complete structure is too big for this screen. Do you see how some of the nodes are collapsed and we are still not able to see both Shadow Hosts, both Shadow DOM's or element in 1 layout? I removed the unnecessary nodes so we can focus on the elements for our Test Script. All of the expanded nodes are highlighted green, red, or gray. Recall from the introduction, a Shadow Host is an element with a Shadow Tree. In this diagram, both Shadow Hosts are green. The Shadow DOM begins with a Shadow Root which is highlighted red. Our nested Shadow DOM is the second shadow-root. Then we see the button element starting with an <a> tag.



Just like in the second session, we can copy the JS Path by right clicking, selecting Copy then Copy JS path. However, in this session, we are going to create the JavaScript Path step-by-step. Go to the Console and clear the console. Know what, I'm going to go ahead and paste it then hit enter.



The JavaScript path starts with document.querySelector and a cssSelector value. This means the element matching body > shop-app will be returned so if I type document.querySelector and the cssSelector ('shop-app'). We see the next output has shop-app as the tag name.



Notice, 2 things about the path I wrote compared to the path that was copied from Elements tab. My path has single quotes and only shop-app as the CSS Selector value. The copied path has double quotes and body > shop-app as the CSS Selector nested values. Either way is okay, I chose to bypass the body

node. If we expand each node then all of the subsequent nodes will make this here look exactly like the Elements tab.

I'm going to collapse the node and go back to the diagram. We see shop-app is the Shadow Host and comes after body. Next, is the shadow-root which let's us know we are at the Shadow DOM. Within the first Shadow DOM is the iron-pages tag followed by the second Shadow Host shop-home. That's all we need for the first Shadow DOM. For CSS Selector, we can skip iron-pages and go directly to shop-home. In the Console, we can click the up arrow button and the up arrow shows the previous statement. To access the shadow root, we write dot .shadowRoot.querySelector('shop-home').

```
document.querySelector('shop-app').shadowRoot.querySelector('shop-home')
▶ <shop-home name="home" class="iron-selected" visible>…</shop-home>
```

shadowRoot represents the shadow root element and is the root for this component. Query Selector works inside of the Shadow Tree. Do you see how we are walking through the DOM step-by-step? In the diagram, we are at the nested Shadow DOM level and it's the same process. For the Shop Now button, I am going to get the shadow-root, bypass the div tag, then return shop-button and the element which starts with the <a> tag. In the Console, we write dot .shadowRoot.querySelector('shop-button > a'). Here's the anchor tag displaying our element.

```
document.querySelector('shop-app').shadowRoot.querySelector('shop-
home').shadowRoot.querySelector('shop-button > a')
  <a href="/list/mens_outerwear" aria-label="Men's Outerwear Shop Now">Shop Now</a>
```

Next we are going to copy the JavaScript path then go back to our code. We have the WebElement which is the shopNowButton =. It's important to convert to a (WebElement) from the jsExecutor.executeScript(" ") method. At this point, we return the JavaScript path. Let me modify the path so it all shows up on the same screen. This is the path we just created.

```
// Find The WebElement So We Can Perform An Action
WebElement shopNowButton = (WebElement) jsExecutor.executeScript(
        script: "return document.querySelector('shop-app')." +
                "shadowRoot.querySelector('shop-home')." +
                "shadowRoot.querySelector('shop-button > a')");
```

First line returns the shadow host from the original DOM. Second line is the shadow host from the nested Shadow DOM. Third line is the Shop Now button. With the shopNowButton, we click().

```
@Test
public void findElementInNestedShadowDOM () {
  // Provide Access To JavaScript
  JavascriptExecutor jsExecutor = (JavascriptExecutor) driver;

  // Find The WebElement So We Can Perform An Action
  WebElement shopNowButton = (WebElement) jsExecutor.executeScript(
          script: "return document.querySelector('shop-app')." +
                  "shadowRoot.querySelector('shop-home')." +
                  "shadowRoot.querySelector('shop-button > a')");
  shopNowButton.click();
}
```

Let's Run. The test passed. We see the button was clicked and the Next page shows up with items.

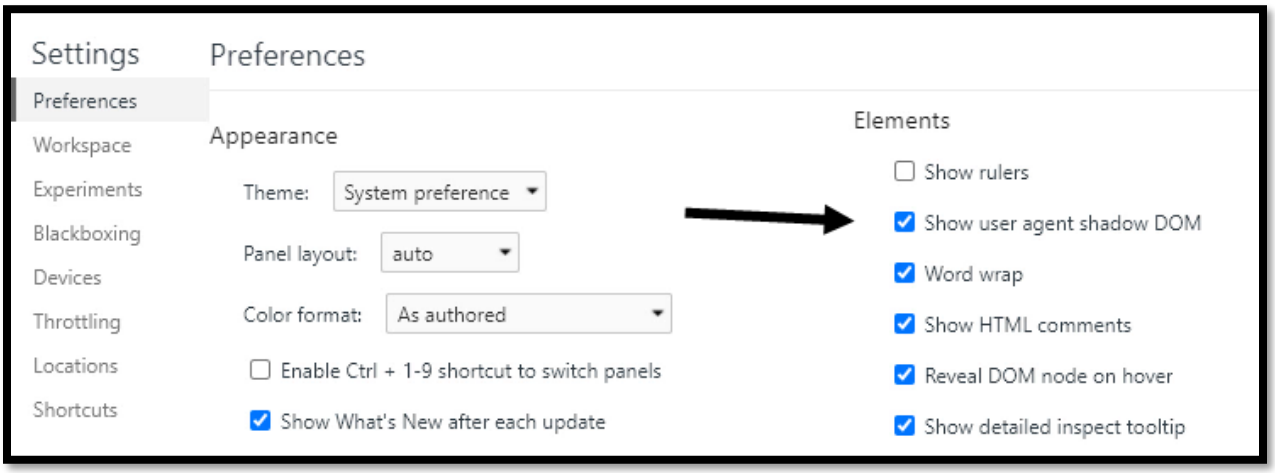
Access Closed Shadow DOM / shadow-root (user-agent)?

In this session, we are going to look at a closed Shadow DOM and answer the question "Are we able to access a closed Shadow DOM?". Straight to the point – No we are not able to access a closed Shadow DOM. However, I'm going to show you why and also explain why we are not able to access a closed Shadow DOM.

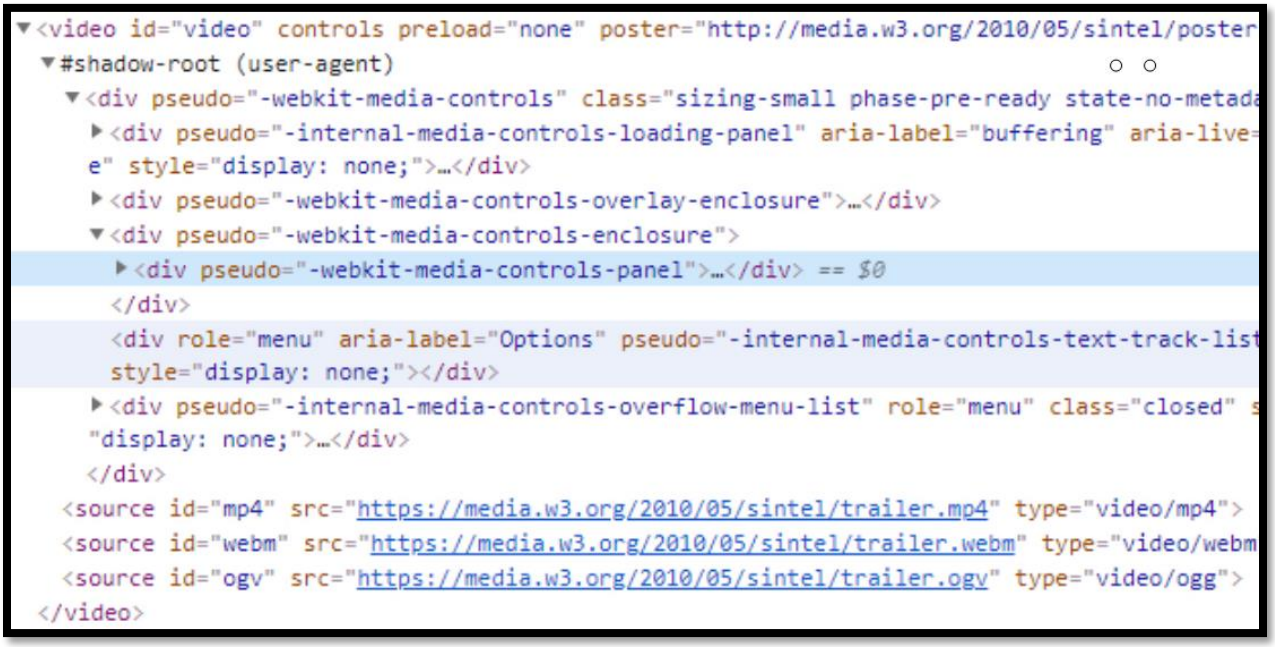
Rob Dodson wrote an article introducing Shadow DOM. He uses this video player as an example.

On the video player, we see a play button that turns into a pause button, time tracker, volume control, a way to make it full screen and a way to Download. On the other hand, if I inspect, all we see is the video tag with some nested source tags.

Where are the other tags such as playing the video. Those other tags are hidden using a Shadow DOM. Notice the breadcrumb does not show shadow-root. But, we can see how the browser implements Shadow Root by going to Settings and clicking the checkbox for Show user agent shadow DOM under the Elements section.

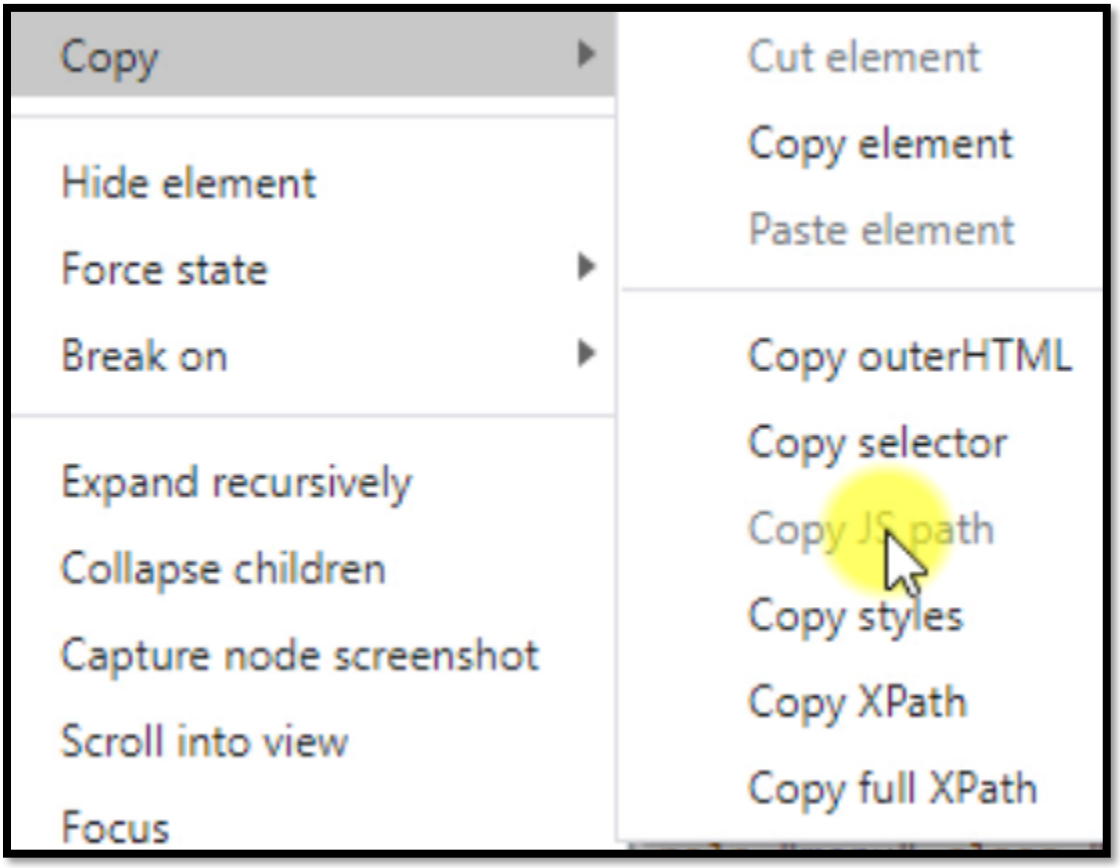


Close the Settings, Reload the page, and inspect the video player again. The video and nested source tags are still here. Now, we also see shadow-root in the DOM and breadcrumb.



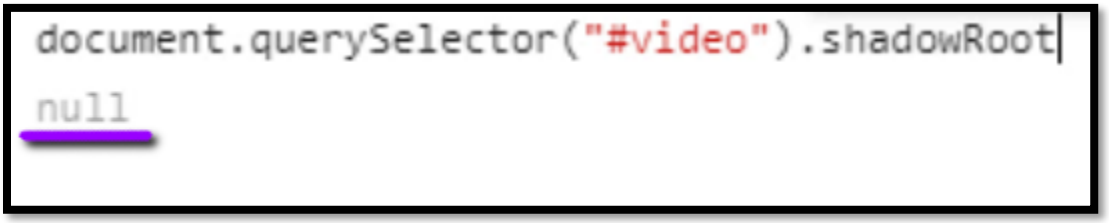
You know how we could not find any tags for playing the video. Watch what happens when I expand the last div tag. We see Play and those other media controls. That's encapsulation because they chose to hide the details. The checkbox "Show user agent shadow DOM" allowed us to analyze the encapsulation. There are 2 types of encapsulation modes: open and closed. They have the same purpose, but the difference is an open Shadow DOM mode allows us to access the Shadow Tree. However, a closed Shadow DOM mode does not allow us to access the Shadow Tree even if we use JavaScript. There is no way around it. This is a closed Shadow DOM because the element shows shadow-root (user-agent).

Recall from the previous session, we can create our own JavaScript Path or copy the JavaScript Path. I'm going to try and copy by right clicking, selecting Copy then Copy JS Path and Copy JS Path is disabled. It does not highlight with a gray background.



Let's go ahead and copy the JS Path for the Shadow Host which is the video tag (Copy – Copy JS Path) and go to the Console. Clear it and Paste. Hit enter. Now we see the video tag. If I maximize, we see the shadow-root. Maximize one more and now we the see div tag that had the media controls.

Access the shadowRoot after clicking the up arrow key .shadowRoot. Do you see null?



Let's finish and write .querySelector('div'). Now I'm going to hit enter. We have an error "Uncaught TypeError: Cannot read property of null".

That means we are trying to access a property of an object that is not defined. Thanks for watching and I will see you in the next session.

Contact Info

- ✔ YouTube https://www.youtube.com/c/RexJonesII/videos
- ✔ Facebook https://facebook.com/JonesRexII
- ✔ Twitter https://twitter.com/RexJonesII
- ✔ GitHub https://github.com/RexJonesII/Free-Videos
- ✔ LinkedIn https://www.linkedin.com/in/rexjones34/