DOM DI

DOM DI is a DOM Abstraction Layer as a Dependency Injection Container. It helps you build JavaScript functionalities that are completely decoupled from the DOM in the spirit of the Separation of Concern (SoC) and Inversion of Control (IoC) design patterns.

# What can a DOM DI container do for you?

The problem is that selectors work with the DOM’s structure and semantics. So when you directly traverse the DOM with some assumed selectors, from within your components, you assume some knowledge of the DOM and bear the responsibility of keeping both sides in sync.

First, ask the component! Sometimes all it really needs is a simple list of elements to tickle; not where they live or how they are made, nor the cost of finding them and maintaining them. That means you can completely keep the DOM out of functional code. And you will be able to prototype rapidly, test with mockups, and progressively evolve your app’s HTML and JS separately without one getting in the way of the other. This is the Separation of Concern (SoC) that a DI container brings.

Second, if your component is heading outdoors, it would be completely impractical to expect, or decide, that every DOM in the wild be structured in some way, just for your component to work. The modern wisdom is that we relinquish control to the code that calls our component and simply accept elements from them instead of hunt elements in a DOM we know nothing about. This is the Inversion of Control (IoC) that a DI container brings.

# How it works!

DOM DI is a DOM Abstraction Layer that accepts a list of selectors, resolves all intricacies and dependencies (when an element depends on another), and simply delivers a flat registry of elements that your code gets to see. With this, a component, for example, just exposes the list of elements it needs and allows the calling code to pass them in as selectors.

# Usage

DOM DI comes in a tiny class that takes a base element and a list of selectors to resolve from this base.

var domdi = new DomDi(‘#my-form’, {

button: ‘#button-1’,

spinner: ‘#ajax-spinner’,

});

To access your elements, use the .registry() method.

var registry = domdi.registry();

All of this is even easier as a jQuery plugin; no need to deal with a physical class.

var registry = $(‘#my-form’).domdi({

button: ‘#button-1’,

spinner: ‘#ajax-spinner’,

});

Now, the rest of our code does not have to worry about the DOM. And we can simply pass the registry object around.

## The Registry