TOM: A New Model for HTML

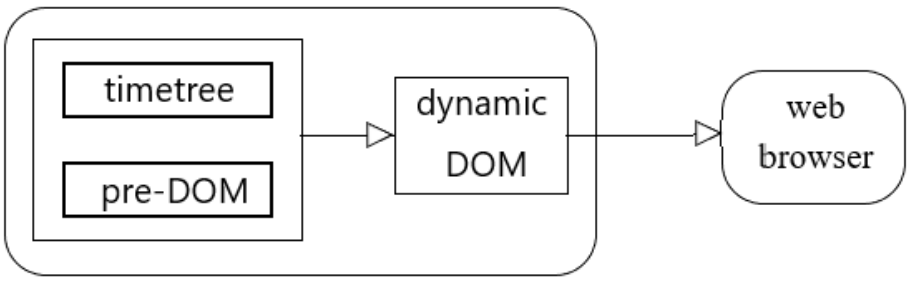
Hui Wang

Today modern web site accelerated by scripts[1], but the foundation, web page[2] its self is still a static structure. Document Object Model[3](DOM) represents the structure of web page. Here we show a new approach: It is possible to put timetree[4] and DOM together to shape a new structure named Time Object Model. TOM represents not only a static page but also a dynamic stream. We believe the best way for using TOM is to embed it into a HTML[5] page in real time without changing the existence, it is the only way works now.

Introduction

DOM and TOM, both of them represent HTML contents, what is the difference? It is: DOM put everything at a single time point, TOM separates its contents in a timetree.

TOM constructed with three parts: The first part is timetree, it describes a period of time. The second part is pre-DOM, it contains all the HTML elements. It looks just like DOM, but it is not. The elements inside pre-DOM are separated into different time, normally not all of them be shown at same time. The third part is a dynamic DOM, it contains all the elements shown at current time.



TOM

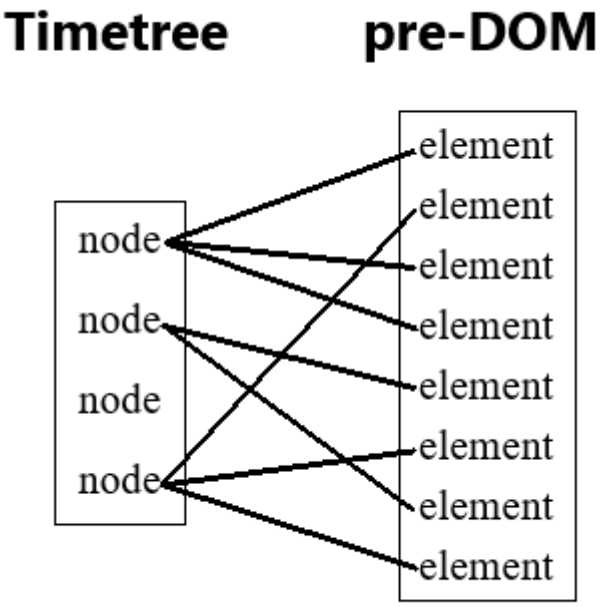
The workflow of TOM is really simple: pre-DOM contains elements all over the time, timetree is a filter helps to pick up the visible elements according to the current time. The elements picked will be reassembled to generate a dynamic DOM. When it is completed the dynamic DOM will be delivered to web browser[6] to show a web page or embedded into another DOM. The steps above represent a single frame of TOM. Repeat the procedure it will show a dynamic web stream.

TOM is the up level of DOM, if we want to render it up, we must convert it to DOM

first. No web browser supports it, we depend on special runtime, it is written by javaScript. The runtime will manage to run TOM up, converts TOM to DOM through the time, then delivers the result DOM to web browser engine continually.

The workflow of TOM

Here we introduce TOM in detail. Inside TOM, timetree and pre-DOM constructed a map [Fig]. Inside map each HTML element is mapped to one and only one timetree node. Node is a container contains time, if an element mapped to a node, that means the node contains the lifetime of element. In the case, we also call the element is hosted by the node. The lifetime of host-node decides what time the hosted-elements be alive. That means the element will be visible if its host-node is visible, will be invisible if its host is invisible.



The pre-DOM is similar to DOM, it is also a hierarchy structure, also contains HTML elements and containers. But the different part is: The members of HTML container live in different time. The HTML container of pre-DOM contains elements which be hosted by different nodes, so its visible contents are changing during time, that will not happen within DOM. Obviously if a container be hosted by a node, its members must be hosted by the same node or its descendants.

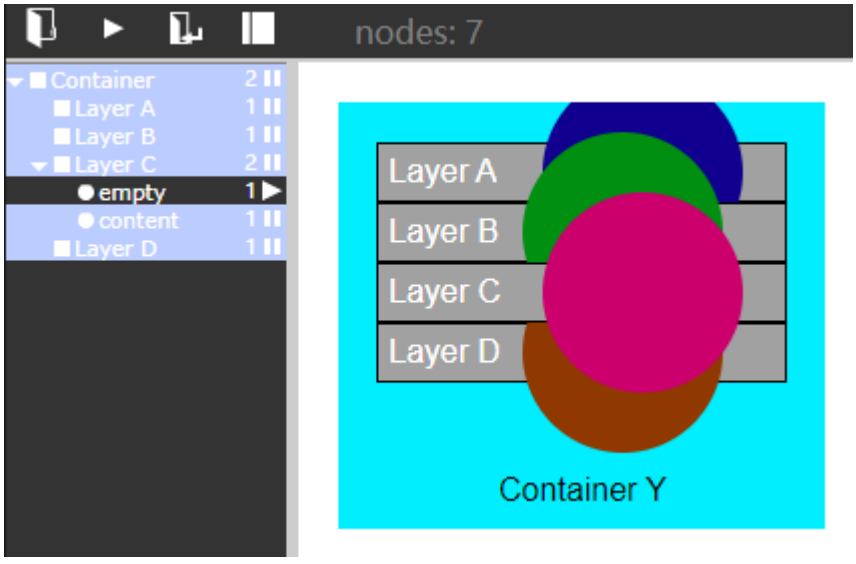
The timetree of TOM is playing. First step of workflow we must pick up all the nodes visible now, then find the elements be hosted by these nodes, finally we got a collection of visible HTML elements. We need to assemble them together to create a dynamic DOM, but their order is still uncertain. The elements hosted by the same node always shown or hidden simultaneously, so their order is fixed. But the elements be

hosted by different nodes, we must link them in dynamic. Here is the method: Each node of timetree has an attribute of “node-order”, the hosted elements inherit the attribute of its host. If a node be ahead of another node, its hosted elements also be ahead of the elements which be hosted by another node.

We know pre-DOM is a hierarch structure, it contains many levels of containers. It is very possible multiple containers be hosted by same node, the key point is: each container has its own setting of “node-order” at the same node. It gives the power of flexibility to each individual container, allows it to customize its own order. It sounds somehow complicated, but it works precisely.

Now the ordered dynamic DOM is on ready, it is time to the CSS[7] part. HTML has its own way to layout DOM up, it works with TOM, we do not discuss it. Here we suggest a new option: We do not use the CSS attribute of “display” and “position”, but use another attribute called “container-order” to replace it. It is only available with HTML container, it has three options: “x-order”, “y-order” and “z-order”. The “x-order” is similar to the “inline” or “inline-block” of attribute “display”, it sets a horizontal array. The “y-order” equivalent to the “block” of attribute “display”, it aligns its members in vertical direction. The “z-order” plays the same role of “z-index”, it aligns elements in deep space but does not need an absolute number.

We still face a challenge, sometime the setting of depth is required even elements are set to x-order or y-order. For example, several elements be set as y-order, they have overflowed inner contents, and the attribute “overflow” be set to “visible”, which one will cover others? We suggest here if no explicit depth setting be there for elements, by default the depth order depends on the loading order of elements, the latest loaded element always be at the top. In the diagram below, Container Y is set as y-order, its member “Layer C” must be on top because it is the latest one added.



The sample of TOM

Here we show a really simple sample of TOM. It is a functional tree with only two nodes, it has two conditions: “open” and “close”.

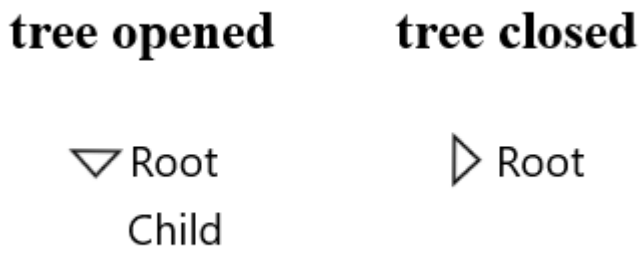
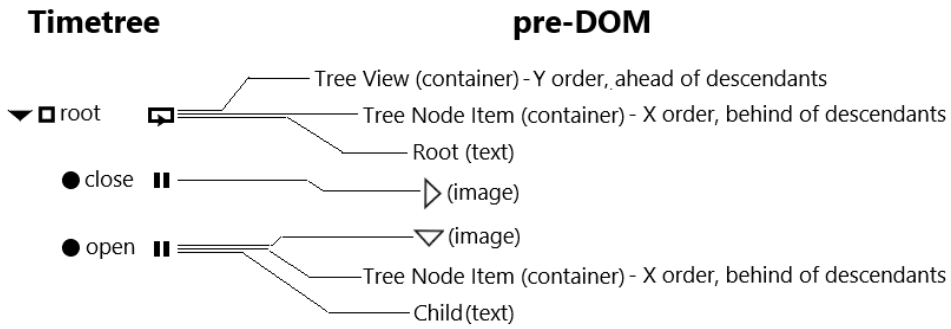
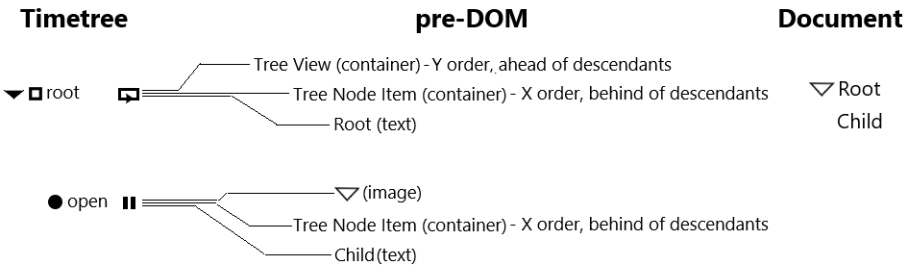


Figure following shows the entire structure of TOM. Timetree has three nodes, each child represents a condition of tree. Pre-DOM contains all the elements over time. These two established a map, inside the map each element of pre-DOM is hosted by one and only one node of timetree.

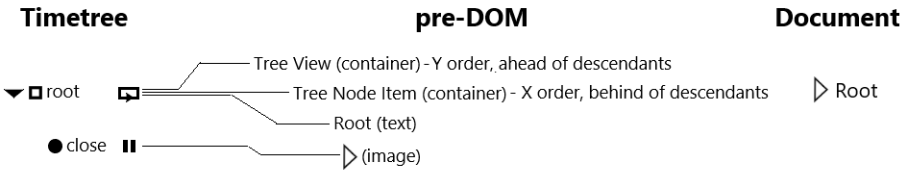


The tree has two conditions. At any condition, first we check out the timetree what nodes are visible. Then find the elements be hosted by them, these are visible elements. The next move, we keep the visible elements in, but remove invisible elements away (or just hide them) from pre-DOM. The remained structure is still not Dynamic DOM, we need to determine the order of elements. Yes, we can set all orders up at the very first time pre-DOM built up, then just show/hide elements, but we still face the same question.

As we discussed above, each container has its own space order at even same host node. In this case Tree View is the top-level container, it is hosted by root of timetree. Its host-node must be set to " ahead of descendants", because we must make sure the parent node always be above with its descendants. The first Tree Node Item is also hosted by root, it is also a container but set its host-node as " behind of descendants", because we want its disclosure icon always be ahead of the text. After the adjusting step, finally we got the Dynamic DOM, then deliver it to web browser for rendering.



Tree opened



Tree closed

The tree must be responsible. TOM is a dynamic model it supports some dynamic methods including:

Jump to. (set the local time of nodes)
Next/previous frame. (set local time to next/previous frame)
Next/previous child. (set local time to next/previous child)
Initialize/Counter-initialize. (set local time to minimum/maximum value)

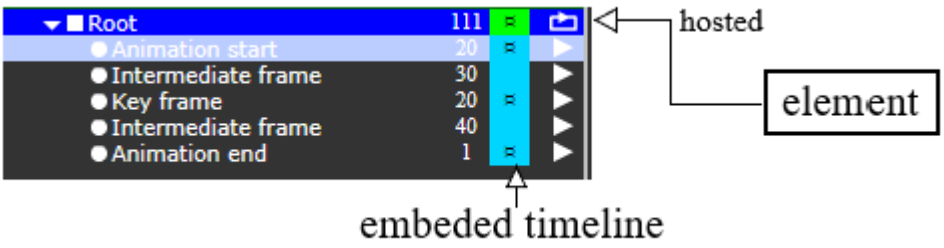
Here we take method “jump to next child”. We make sure when the disclosure icon got clicked, its host-node always jumps to its next child. The method is flexible, it works at any tree level. Now the tree becomes alive.

In reality, normally pre-DOM is just a logic structure for describing the relationship between elements but without existed because it is not necessary. Only DOM is visible to users, so we just create DOM. We keep the information of pre-DOM in convenient way but do not create it.

The animation of TOM

If the host-node of element has descendants, we always have chance to change element's CSS attributes during its lifetime, it will create CSS animation[8]. We use embed-timeline to represent its animation [Fig]. The timeline located along side with timetree, we are able to insert keyframes inside it if the host has enough descendants. The CSS

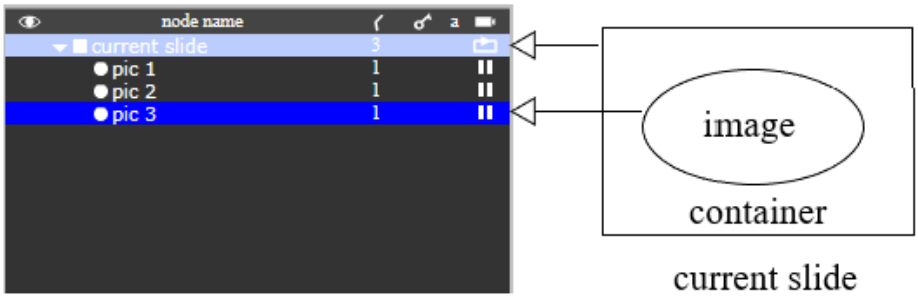
keyframe is hosted by one and only one node, similar to HTML element. Making animation with TOM is very easy: first pick a HTML element, then select a node inside its host, finally change a CSS attribute to create a keyframe, that is it.



If the host node of element has parallel descendants, things will become very interesting. We know parallel nodes existed simultaneously, so the timeline-clips hosted by them must be playing at same time. We never see it happening with a timeline, what is the result? The result is: TOM will merge them together in real time, this is Independent Animation. Independent Animation allows us combine animation in dynamic way. Inside its bundle, each one is independent, whatever we change, edit, insert or remove anyone, the others will not get effected.

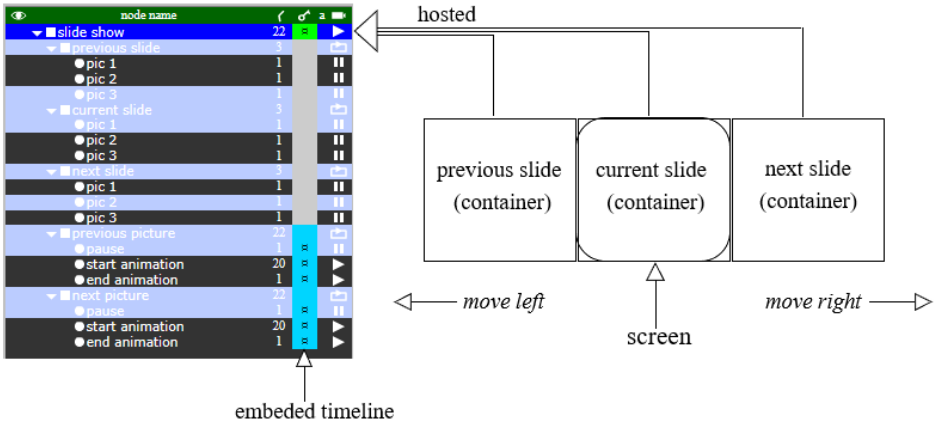
If we want to add an Independent Animation for a HTML element, we just need to make sure two requirements got met: First the animation must be inside a parallel node, second the parallel node must be inside the host-node of target element. Let's show you through samples.

Slideshow of image is very common component on internet, let's make it. First, we make a really simple one [Fig]: We put navigate buttons on node "current slide", each child hosts a picture. "current slide" will go to previous or next frame when the buttons got clicked. Really simple, but without animation.



It is time to add animation effects now, here is our plan [Fig]: We use five parallel modules to reach the goal. Each one of them focus on one simple task, the final result depends on the combination of these five. "current slide" always represent the current picture. "previous slide" and "next slide" are the clones of "current slide", they always show the previous/next picture according to the current picture. "next picture" always shows the dynamic effect of "move forward", "previous picture" is similar but move backward.

Independent Animation



We have five functional modules now, but still need to meet the second requirement as we discussed above. We wrap each slide element with a new HTML container element, the container be hosted by root, and root includes all these five. Now every requirement is met. We animate the container, not the pictures. It is more flexible, because now we do not care about how many pictures the container contains, the component is easier to reuse.

Thanks to the independency, now we are able to build more complicated architectures to support more complicated animations. The Independent Animation also called Structured Animation, it constructed from a hierarch structure more complicated than a liner timeline.

The practice of TOM

We still face a question, how to use TOM? How many costs it will take if we using TOM? Very tiny. If you want to use TOM, just do the following.

1. Load the runtime first (tom.js) in your HTML.

2. Prepare a HTML container element for loading TOM, so far only tag got tested.

3. We saving TOM data as JSON[9] format data, so then call the runtime API to load a JSON file/data from somewhere through the tech of AJAX[10].

When downloading completed, the runtime will parse data automatically, then runs it up.

When we embed a TOM into a HTML, in fact it is the Dynamic DOM of TOM be inserted into the DOM of target HTML. Do not modify the Dynamic DOM clip, let runtime do its job. If you do not want to control it through javaScript, that is all until you want to remove it or replace it. If you want to use javaScript, there has two options: you can call runtime API directly to control the play of TOM, also you can capture the DOM event raised from TOM. So far TOM still does not support real time editing, only you can do is control its play or receive its event information. If you want to create TOM, you

need a special software[11].

Conclusion

Script is a reliable way for operating TOM just like DOM. But here we suggest another way, maybe we can manipulate TOM without script. The way of non-script is functional limited on many sides, and also efficient less. But only we need is just make it be a bridge to connect with users and computers. It is hard, but worth to try. Because it will increase everyone's capability, and it is fundamental.

Reference:

[1] "JavaScript", https://developer.mozilla.org/en-us/docs/web/javaScript

[2] "What is Web Page? -Computer Hope", https://www.computerhope.com/jargon/w/webpage.htm

[3] "DOM Living Standard", https://dom.spec.whatwg.org

[4] "Timetree: A New Way for Representing Time", https://vixra.org/abs/2401.0124

[5] "HTML Living Standard", https://html.spec.whatwg.org/multipage/

[6] "What is a Web Browser?", https://www.mozilla.org/en-us/firefox/browser/what-is-a-browser/

[7] Cascading Style Sheets Level 2 Revision | (CSS2.1) Specification, https://www.w3.org/TR/CSS21/

[8] "The Ultimate Guide to Animations in CSS", https://blog.nubspot.com/website/css-animation

[9] "JSON", https://www.json.org

[10] Rahui Awati, "AJAX (Asynchronous JavaScript and XML)", https://www.theserverside.com/definition/Ajax-Asynchronous-JavaScript-and-XML

[11] "TOMStream", https://store.steampowered.com/app/2495810/TOMStream