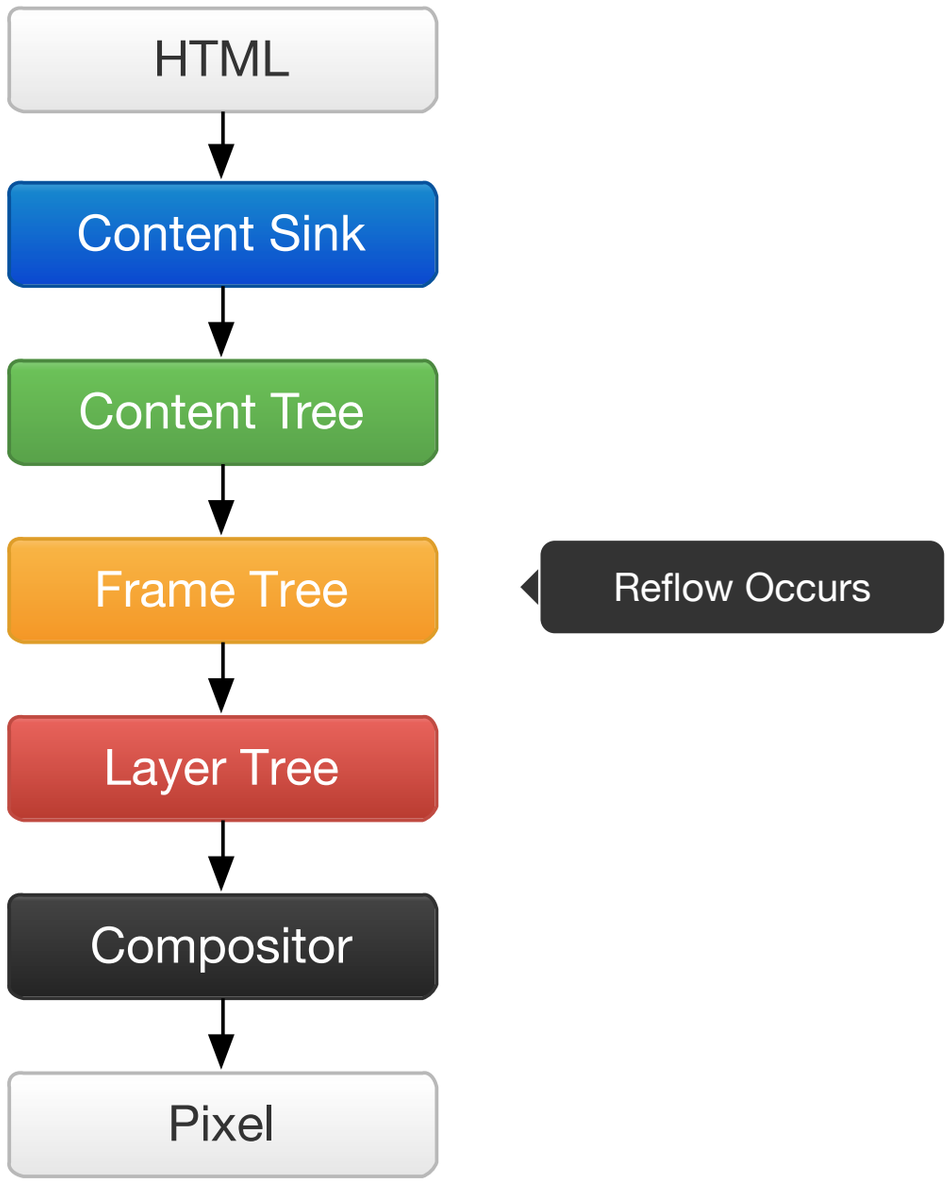
Graphics Subsystem Overview

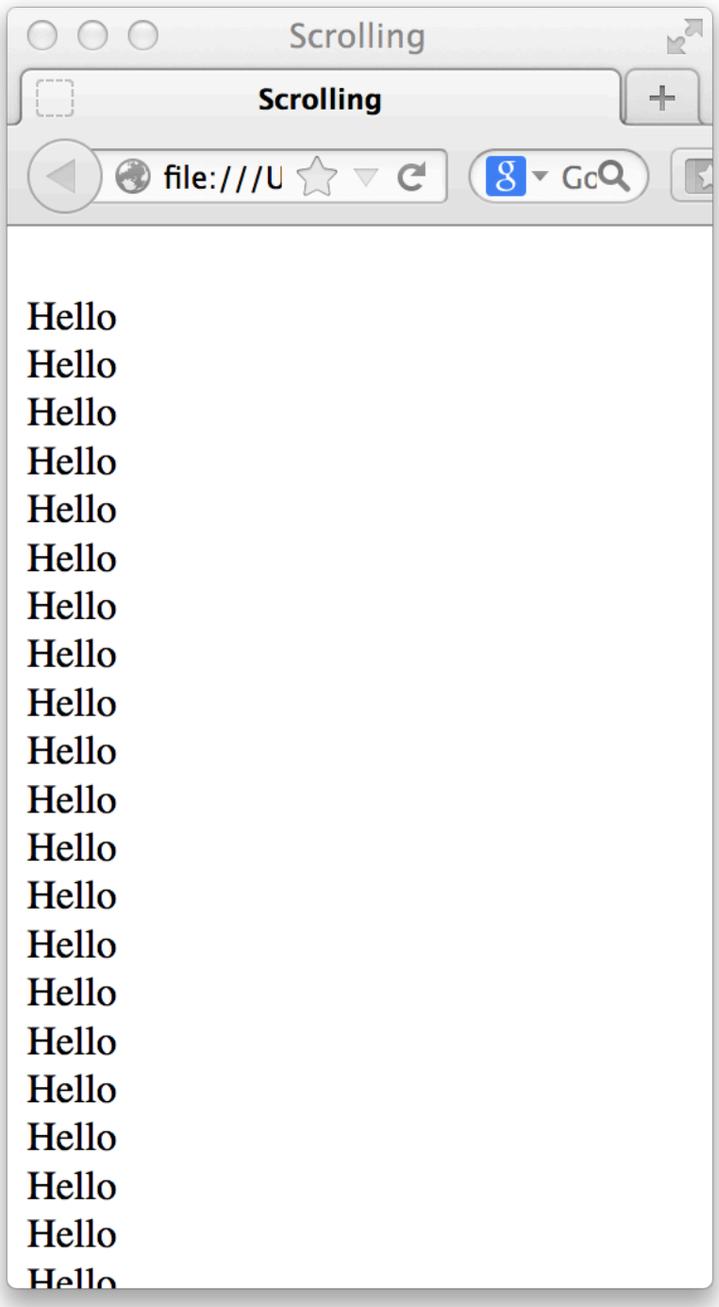
(Ignoring CSS for Now)

Mason Chang

Architecture



Simple Example




```
<html>
  <head>
    <title>
      Scrolling
    </title>

    <script type="text/javascript">
      function load() {
        var elem = document.getElementById("test");
        var text = "";
        for (var i = 0; i < 1000; i++) {
          text += "<br>Hello";
        }

        elem.innerHTML = text;
      }

    </script>
  </head>
  <body onload="load()">
    <div id="test"> </div>

  </body>
</html>
```

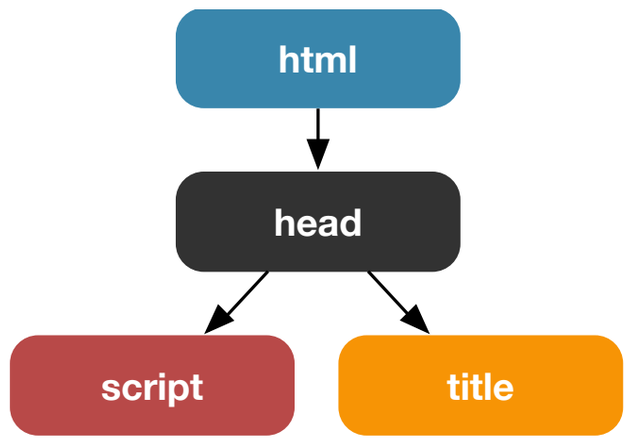
Content Sinks

- Parses Input HTML and generates Content Tree, mostly 1:1 between HTML and Content Node. Like an Abstract Syntax Tree (AST) in compilers

```
<html>
  <head>
    <title>
      Scrolling
    </title>

    <script type="text/javascript">
```

Translation →



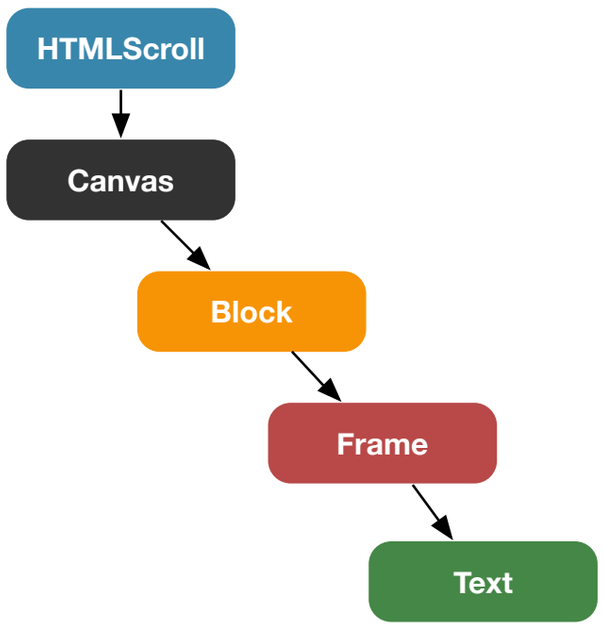
Content Tree

- Layout Debugger -> Dump Content Tree

docshell=0x10c807120
html@0x10cb21f00 state=[40000010000] flags=[00400008] primaryframe=0x10cb96480 refcount=10<
head@0x10cb21fc0 state=[40000010000] flags=[00400000] primaryframe=0x0 refcount=7<
Text@0x10cb27940 flags=[00000000] primaryframe=0x0 refcount=1<\u000a >
title@0x10f8c0920 state=[40000010000] flags=[00400000] primaryframe=0x0 refcount=2<
Text@0x10cb27b80 flags=[00000000] primaryframe=0x0 refcount=1<\u000a Scrolling\u000a >
>
Text@0x10cb27c10 flags=[00000000] primaryframe=0x0 refcount=1<\u000a\u000a >
script@0x10c710d10 type="text/javascript" state=[40000010000] flags=[00400000] primaryframe=0x0 refcount=2<
Text@0x10cb27dc0 flags=[00000000] primaryframe=0x0 refcount=1<\u000a function load() {\u000a
var elem = document.getElementById("test");\u000a var text = "";\u000a for (var i = 0; i < 1000; i++) {\u000a
text += "
Hello";\u000a }\u000a\u000a elem.innerHTML = text;\u000a }\u000a\u000a > - JavaScript
>
Text@0x10cb27e50 flags=[00000000] primaryframe=0x0 refcount=1<\u000a >
>
Text@0x10cb27ee0 flags=[1e000000] primaryframe=0x0 refcount=1<\u000a >
body@0x10f8c0a60 onload="load()" state=[40000010000] flags=[00400000] primaryframe=0x10cb96cb8 refcount=5<
Text@0x10cb880d0 flags=[1e000000] primaryframe=0x0 refcount=1<\u000a >
div@0x10f96b520 id="test" state=[40000010000] flags=[00400000] primaryframe=0x10cb96d90 refcount=2002<
br@0x10f96b8e0 state=[40000010000] flags=[00400000] primaryframe=0x10cb97bd8 refcount=2<>
Text@0x10cb88550 flags=[0c000000] primaryframe=0x10cb97c28 refcount=2<Hello>
br@0x10f96ba20 state=[40000010000] flags=[00400000] primaryframe=0x10cb97c98 refcount=2<>
Text@0x10cb885e0 flags=[0c000000] primaryframe=0x10cb97ce8 refcount=2<Hello>
.....
br@0x10f96bb60 state=[40000010000] flags=[00400000] primaryframe=0x10cb97d58 refcount=2<>
Text@0x10cb88670 flags=[0c000000] primaryframe=0x10cb97da8 refcount=2<Hello>
br@0x10f96bca0 state=[40000010000] flags=[00400000] primaryframe=0x10cb97e18 refcount=2<>
>
Text@0x10cb88280 flags=[1e000000] primaryframe=0x0 refcount=1<\u000a\u000a \u000a\u000a>
>
>

Frame Tree

- In Layout Debugger -> Dump Frame Tree
- Mostly 1:1 Translation between Content Tree to Frame Tree
    - initialized in layout/generic/nsFrame*::Init for frames
    - PresShell::ContentAppended - When Content Sinks find new content to append to the Frame Tree
- layout/generic/nsFrame.h - Associated with a content frame sometimes:
    - nsFrame::GetContent -> returns nsIContent



Example Frame Tree

webshell=0x114f5f920

Viewport(-1)@0x121a59420 [view=0x120972900] {0,0,59400,54840} [state=0000062000002220] [sc=0x121a590f8:-moz-viewport]<

HTMLScroll(html)(-1)@0x121a5a0a8 {0,0,59400,54840} [state=0000020000000010] [content=0x1125c5d40] [sc=0x121a59758:-moz-viewport-scroll]<

ScrollbarFrame(scrollbar)(-1)@0x121a5a830 next=0x121eb4ba0 {0,54840,59400,0} [state=0000100080c80000] [content=0x11409e620] [sc=0x122557a28]<

SliderFrame(slider)(-1)@0x121eb4280 {0,0,59400,960} [state=0000160080c00000] [content=0x11409ec50] [sc=0x122557978]<

ButtonBoxFrame(thumb)(0)@0x121eb4718 {60,120,59280,780} [state=2000160080400000] [content=0x11409ed70] [sc=0x122557718]<>

ScrollbarFrame(scrollbar)(-1)@0x121eb4ba0 next=0x121eb5dd0 {59400,0,0,54840} [state=0000100080880000] [content=0x11409e6b0] [sc=0x121a5ac78]<

SliderFrame(slider)(-1)@0x121eb5460 {0,0,960,54840} [state=0000160080800000] [content=0x11409f1f0] [sc=0x121a5a980]<

ButtonBoxFrame(thumb)(0)@0x121eb5868 {120,60,780,54720} [state=2000160080000000] [content=0x11409f280] [sc=0x121a59b10]<>

Box(scrollcorner)(-1)@0x121eb5dd0 next=0x121a59de0 {59400,54840,0,0} [state=0000100080c00200] [content=0x11409e7d0] [sc=0x121eb5a60]<>

Canvas(html)(-1)@0x121a59de0 {0,0,59400,54840} [state=0000006000000210] [content=0x1125c5d40] [sc=0x121ee2020:-moz-scrolled-canvas]<

Block(html)(-1)@0x121ee2480 {0,0,59400,13632} [state=0000100000d00210] [content=0x1125c5d40] [sc=0x121ee23d0]<

line 0x121ee2d50: count=1 state=block,clean,prevmarginclean,not impacted,not wrapped,before:nobr,after:nobr[0x248] bm=480 {480,480,58440,12672} vis-overflow=420,480,58500,12672 scr-overflow=480,480,58440,12672 <

Block(body)(2)@0x121ee2cb8 {480,480,58440,12672} vis-overflow=-60,0,58500,12672 scr-overflow=0,0,58440,12672 [state=0000120000100200] [content=0x121aa4660] [sc=0x121ee2a70]<

line 0x121ee2e28: count=1 state=block,clean,prevmarginclean,not impacted,not wrapped,before:nobr,after:nobr[0x248] {0,0,58440,12672} vis-overflow=-60,0,58500,12672 scr-overflow=0,0,58440,12672 <

Block(div)(1)@0x121ee2d90 {0,0,58440,12672} vis-overflow=-60,0,58500,12672 [state=0000120000100210] [content=0x11409f670] [sc=0x121ee2b10]<

line 0x122557390: count=1 state=inline,clean,prevmarginclean,not impacted,not wrapped,before:nobr,after:linebr[0x8100] {0,0,1,1152} <

Frame(br)(0)@0x121ee3bd8 next=0x121ee3c28 {0,0,1,1152} [state=0000000000000200] [content=0x11409fa60] [sc=0x121eb44b0]

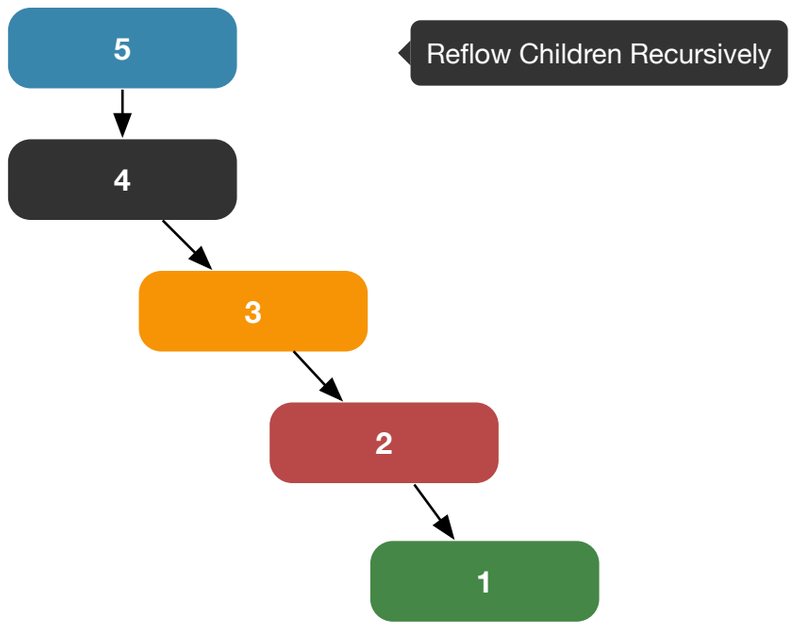
>

line 0x1225573d0: count=2 state=inline,clean,prevmarginclean,not impacted,not wrapped,before:nobr,after:linebr[0x8100] {0,1152,2131,1152} vis-overflow=-60,1152,2251,1152 scr-overflow=0,1152,2131,1152 <

Text(1)"Hello"@0x121ee3c28 next=0x121ee3c98 {0,1248,2131,960} vis-overflow=-60,0,2251,960 scr-overflow=0,0,2131,960 [state=0000004090600000] [content=0x11409faf0] [sc=0x121ee2bc0:-moz-non-element] [run=0x0][0,5,T]

Reflow

- Calculates the width / height of frames - Go through the frame tree and calculate the width/height of each child recursively.
- layout/generic/nsFrame*::Reflow
- Can track reflows with pref - **devtools.webconsole.filter.csslog.** Or Enable CSS:LOG preference in Web console
- Biggest misconception - Even if nothing changes, we sometimes still have to reflow because we have to calculate the sizes of things. (background DOM manipulation)
- Inverse also true, just because some things are changing doesn't mean we recalculate sizes of elements. Scrolling is a great example.



Frame Tree -> Layer Tree

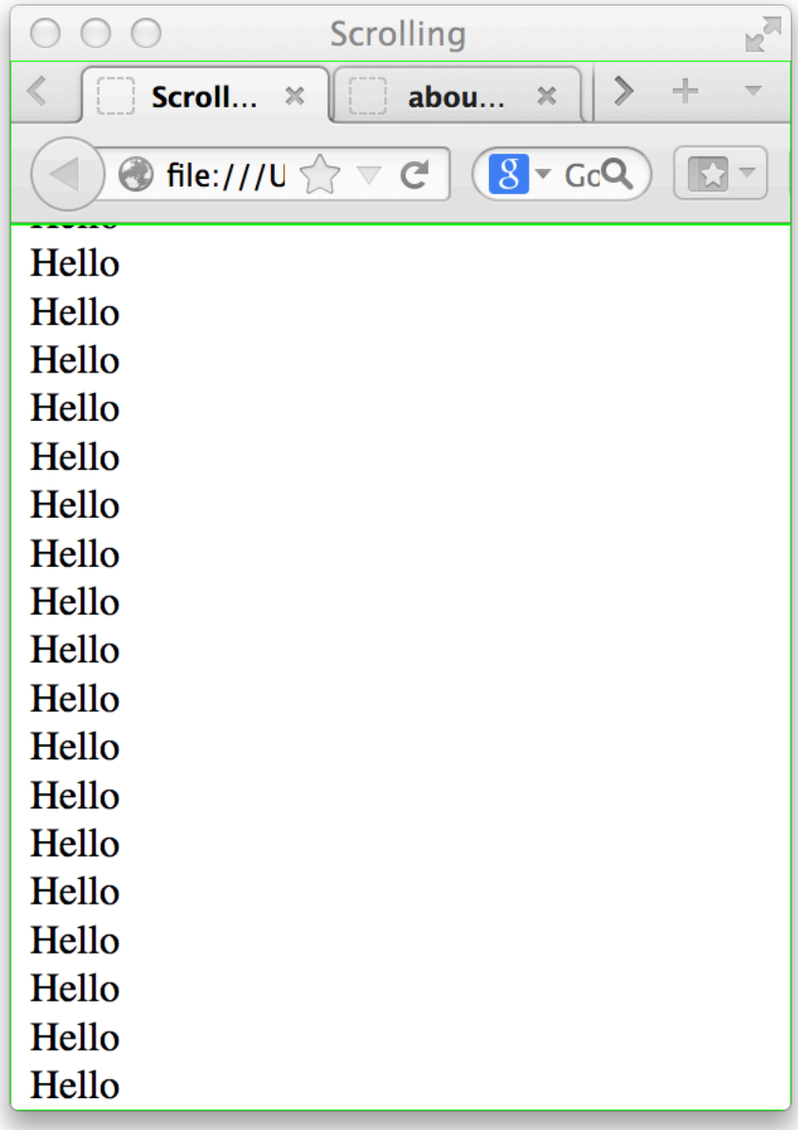
- Occurs at
  “Painting” time - nsLayoutUtils::PaintFrame

- nsFrame*::BuildDisplayList - Determines which frames go into which layer.

- This is the ‘Layout’ portion in a Cleopatra profile.

Layer Tree

- Visualize by enabling **layers.draw-borders**
- Can see all layers by enabling **layers.dump**



Painting the Layer Tree

- Multiple layers, issues drawing commands onto each layer depending which frames are there. Drawing commands use the moz2d library. The moz2d library wraps around 2d drawing backends like Skia / Cairo.

- Drawing commands are higher level than OpenGL. Things like ‘ render this text’, ‘draw a line’.

- Layout tries to figure out and reuse as much of the layer tree as possible, so heuristics are used to optimize painting and rebuilding the layer tree.

- For example scrolling, Two layers - one for background, one for scrolling.

- Painting each layer is the ‘Rasterize’ portion of a Cleopatra profile.

- Async Pan/Zoom draws more than what you currently see.

Compositor

- Input a painted Layer Tree, flattens out into a 2d image of pixels - Where you can finally see OpenGL commands!
- Compositor should be done within 15 ms
    - Uncomment **#define COMPOSITOR_PERFORMANCE_WARNING - gfx/layers/ipc/CompositorParent.h** - Will warn if Compositor takes more than 15 ms
- Realistically, should be done ~8ms on an otoro. Compositor::BeginFrame -> Compositor::EndFrame timings.

Useful Tools

- layers.dump - True
- layers.draw-borders - true
- Layout Debugger - https://developer.mozilla.org/en-US/docs/Layout_Debugger
- Compositor*::DrawFPS - FPS vs TXN FPS
- Reflow Counter - devtools.webconsole.filter.csslog
- Profiler: ‘-p b2g -t Compositor’ - Can see frames

Performance Warnings

- Enable Draw Layers - If you see lots of layers, probably something bad is happening
    - Layers get ‘flattened’ and cleaned based on a timer. For example, settings app is 5 ms.
        - **will-animate: scroll** CSS property keeps layer active and painting alive
- Watch for things being painted, if it gets a different color and wasn’t scrolled out, double painting - bad.
- 3rd number on Show FPS - Painting pixels too many times, should be roughly ~100. A number of 200 means on average, a pixel is being painted twice. 300, 3x, etc. 300 is really bad, 200 is ok, 100 is ideal.