Browser Bugs & Validation

SIPB Introduction to Web Design
Wednesday, January 20th

Jonté Craighead & Cathy Zhang

Lecture Overview

- A Brief Review: Layout Design
- Browser Bugs
- Validation
- Accessibility
- Site Infrastructure & Organization

The Rendering Issue

- W3C produces specifications for CSS
  - Defines valid properties and values
  - Defines how properties/values affect final rendering
    - Does not specify how to generate rendered page
      - Each browser implements own rendering methods
      - ➡ Potential for rendering errors
- CSS 2, recommended in 1998 has just recently been fully supported

As we have previously discussed, the W3C produces specifications for CSS (like HTML). This specification defines specific properties and their possible values as well as how those properties should alter the rendering of elements on the screen. However, it does not specify how a browser or other rendering application should process the properties and values to calculate the rendering on the screen. Because of this, and the fact that there is no commonly used rendering system or “engine”, each browser is left to design their own methods for interpreting properties and values and translating them into the rendered product. This breakdown in the standardization of the rendering pathway is what allows rendering errors to be possible-- when there are several implementations of the same thing, each will likely be different and be prone to different problems.

As a concrete example of how true this idea is, note that CSS 2, the current CSS specification which was released in 1998, has just recently been fully supported by all major browsers. In essence, it took more than ten years for the major browsers to independently design rendering engines that fully complied with the W3C standards.

In all cases, because the rendering engine is not standardized, there is also the possibility for proprietary/engine-specific properties or values only parsed and understood by those engines (these properties and values do not validate).

Rendering Engines

Trident

Gecko

WebKit

Presto

Under the hood, each browser runs a rendering engine-- the names here list the rendering engine used by each of the most popular browsers.

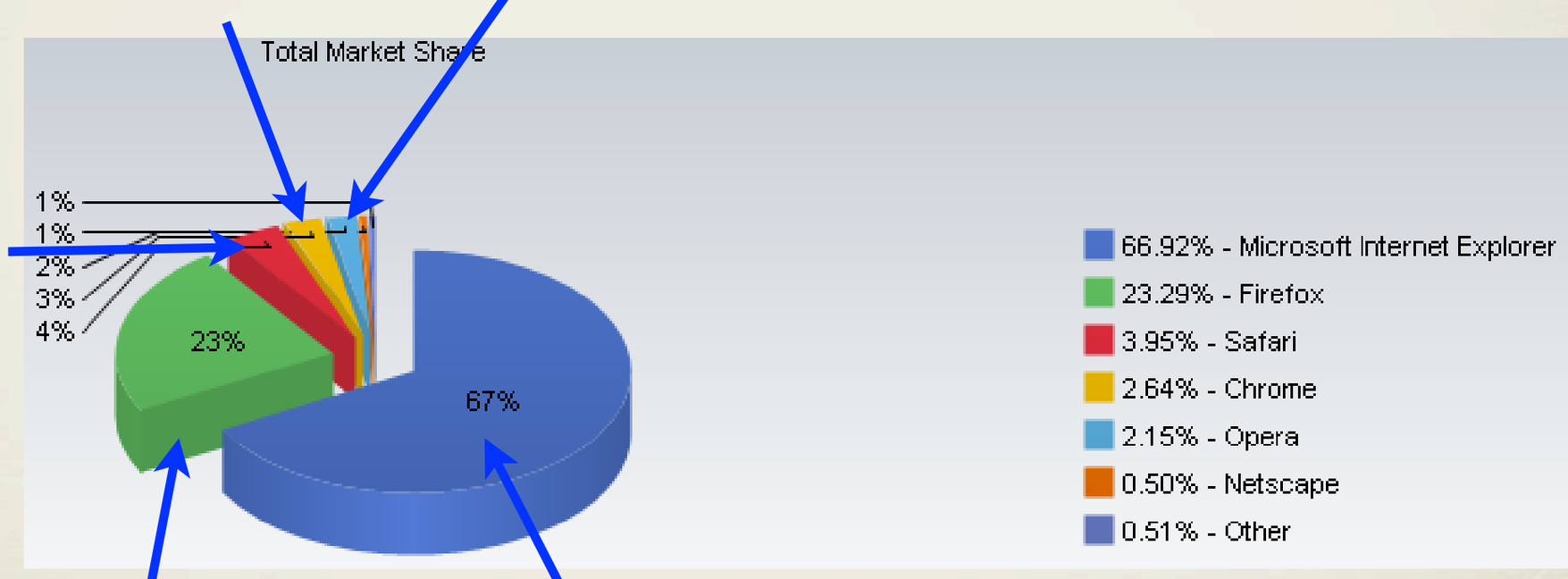
From left-to-right, top-to-bottom, Internet Explorer's rendering engine is named "Trident" and is closed-source; Firefox uses the "Gecko" engine; both Google Chrome and Safari use "WebKit"; Opera users the "Presto" rendering engine. The names of these are not the most important things, but it is useful to note that both Google Chrome and Safari use the same rendering engine.

Browser Market Share

#4

#5

#3



#2

#1

Source: Net Applications

This pie chart demonstrates the market share of the major browsers over the past year (2009). Internet Explorer holds most of the market, while Firefox owns a significant portion at 23%. The WebKit browsers collectively makeup over 5% of browser market share.

A Bit of History

* With IE 6, Microsoft won the browser wars
  * Microsoft allowed IE development to lapse
  * IE pre-installed on Microsoft Windows
    * ➡ Large default market share
* Firefox
  * Developed as open-source alternative from Netscape
* Webkit
  * Open-source engine used by Safari & Google Chrome

When the web was in its infancy, Internet Explorer and Netscape Navigator were the web's most popular browsers. By the release of IE 6, Microsoft had decisively won the browser wars and allowed Internet Explorer development to lapse. Before this time, IE had been the innovator in the field, being more standards compliant than Netscape. Furthermore, with Internet Explorer not only coming standard in the Windows Operating System (IE6 in Windows XP), but actually being integrated into the operating system, Microsoft had no impetus continue development of its browser.

In the innovation vacuum that followed, other new and less-popular browsers such as Opera and the open-source Firefox browsers (rising from the ashes of Mozilla & Netscape) began to become more attractive as they began to support more and more of the W3C's CSS standards. These browsers were especially popular among the technically elite sector of web developers. This holding pattern remained for several years and Internet Explorer continued to lag behind.

Prior to Microsoft's win with IE 6, Microsoft had been continually releasing new versions of IE. Between IE 6 and IE 7, an interval of almost 5 years passed without a major update or release of new features. By this time, browsers such as Firefox were gaining considerable chunks of market share as their use was no longer limited to highly technical audiences.

Recently, within the past two to three years, the open-source Webkit project developed from the Konqueror rendering engine (a linux browser) and gained prominence through its use by Apple's Safari and, more recently, Google's Chrome.

One speculative problem with Microsoft's Internet Explorer, when compared with the open-source alternatives, is that the rendering engine is closed-source. When attempting to address rendering problems, it much like placing different types of input into a black box and seeing what happens to comes out.

Internet Explorer Bugs

- Expanding Box Problem
  - Width: Outer box will always expand to fit contents
  - Height: Outer box's height acts as minimum height
- Float Model Problem I
  - Non-floated content beside float does not flow
- Float Model Problem II (a.k.a "Escaping Floats")
  - Incorrect rendering of multiple floats

The two parts of the box problem have to do with Internet Explorer 6's calculation of width and height attributes. First, for width, the W3C specifies that non-breakable text (text that has too long of a continual length) unable to be completely contained within an element of defined width should extend outside the bound of the container; the container should remain the specified width. In Internet Explorer, however, non-breaking text causes the container to resize.

Why is this a problem? In the case that a layout is created with a sidebar and an explicit width is given for both the sidebar and the content area, the new, larger width will cause the entire width of the layout to be larger than intended and potentially cause the page to extend, creating a horizontal scrollbar.

The second expanding box problem occurs when a box is much taller than the container and the container is given an explicit width. Instead of allowing the contained element to overflow the boundaries of the container (i.e. the container remains its pre-defined height), IE extends the height of the container to fully contain the element. In this way, it is clear that IE treats the defined for an element as a minimum height that it may easily violate.

Internet Explorer has many float problems; Internet Explorer 7 fixed some, but not all of these problems. One of these is noticeable when non-floated (static) content of defined width is placed in the vicinity of a float. In this case, Internet Explorer will render the static content side-by-side with the floated content as if the static content were floated itself. Since static content is supposed to flow around the float and the static content's box begins and ends at either side of its container, static content of defined width is likely set for the entire width of the container. When placed beside a float of defined width, this will cause the static content to extend past the bounds of its container, likely causing a horizontal scrollbar.

Additional, when multiple items are floated and the floats would extend past the edge of the container of defined width, the floats should be wrapped. Internet Explorer not only extends the width of the container as much as possible, it also incorrectly draws a border around the container.

Dealing with Browser Bugs

* Ensure that your markup and CSS are valid
* Develop in a standards-compliant browser
    * Test in other compliant browsers
* Test last in IE, make changes as appropriate
    * Conditional comments to apply IE-specific fixes
* Avoid using Javascript to determine browser

Browser bugs can exist on any browser. The point of detailing bugs in Internet Explorer was twofold: first to provide a concrete example of browser bugs in a browser and to illustrate the danger of developing first in Internet Explorer (and perceiving IE's rendering as the standard). The first step in dealing with browser bugs is to ensure that both your HTML and CSS are valid. When the document does not validate, all bets of cross-browser support are off: the browsers will have to guess what your markup was intending to do and the browser will likely not guess the same things.

Second, develop first in a standards-compliant browser (which now means not IE, that may change in the future) then in other standards-compliant browsers. From experience, if things are inconsistent between standards-compliant browsers, there's likely some changes that can be made in your markup or CSS to fix them.

Always test last in IE; get your layout to look as it should in standards-compliant browsers and use conditional comments to adjust for irregularities in Internet Explorer.

With the exception of a very small number of cases, there is a relatively little reason to use Javascript to detect the browser to simply render your page correctly/supply hacks to make your layout work.

Conditional Comments

- Proprietary IE method for conditionally parsing content
- Target code is wrapped within a comment block
  - Parsed by browser based on condition given
  - Can be used to include non-valid or IE-specific CSS
- Included in comments
  - ➡ Ignored by other browsers
  - ➡ Valid markup & CSS (Ignored by validator)

Conditional Comments as a method is not a technique specified by W3C. It is a proprietary method for causing IE to conditionally parse content in an HTML page. The code to be conditionally executed is fully contained within a comment block containing the condition and the code to be parsed.
If the conditional is evaluated to be true, the code will be parsed, otherwise the code will be ignored. The real power of this feature is that it allows for the inclusion of IE-specific CSS. Because it is included in the comments, other browsers will completely ignore the content resulting in valid markup and CSS (it is also ignored by the validator).

Conditional Comments

A Basic Example

```
<!-- [if IE]>
  <strong>Why are you using IE?</strong>
<![endif]-->
```

A Practical Example

```
<!-- [if IE 6]>
  <link rel=”stylesheet” href=”ie6.css” />
<![endif]-->
```

➡ Include stylesheets only for IE (or a specific IE version)

Now, to see some actual code. The first example checks only to see if the browser is Internet Explorer. If so, the browser will render the tag which is between the if and endif lines.

A much more practical example is one where the browser is questioned as to whether it is IE and version 6. Based on the response, a CSS file could be loaded perhaps containing non-valid CSS declarations to fix rendering problems specific to IE 6.

Validation

* Why validate?
  * Ensures document is well-formed and functional
    * ➡ Best starting point for cross-browser support
  * Enables potential forward-compatibility
  * Assists in finding document errors
* How?
  * Using official W3C validators

Why should you validate your documents? The canonical reason is that it will ensure that your document is well-formed and fully functional according to the specifications in the standard. Because of this it is the best starting point for cross-browser support: browsers strive (or should) to support the standards. Also, it ensures that your document will be forward-compatible: in many years from now, if you have a document valid according to a specific web standard, it will be possible for others to understand and parse your document. Lastly, and perhaps the most important among all of these reasons, it helps you to find errors in your document that you might have overlooked.

How do you validate? You use the official W3C validators.

W3C Validation Service

- Validators for both HTML & CSS
- Allows for validation by link, upload, and direct input
- Provides list of validation errors and suggestions
  - Page is not valid even if there is only one error
- Links to validation results can be included on pages

The W3C maintains validators for HTML & CSS both of which allow you to validate by supplying a link to a document on the internet, through the upload of a document, or through direct input of document code. Once the validation has been run, the validators provide a list of validation errors and suggestions on how they might be addressed. Note that a document is not valid even if there is only one error.

Once a page is on the internet, links to live validation results can be included on pages. This works by having the validator determine the page that you just came from (before clicking the link) and validating that page.