Web Technologies

Lecture 7

Synchronous vs. asynchronous

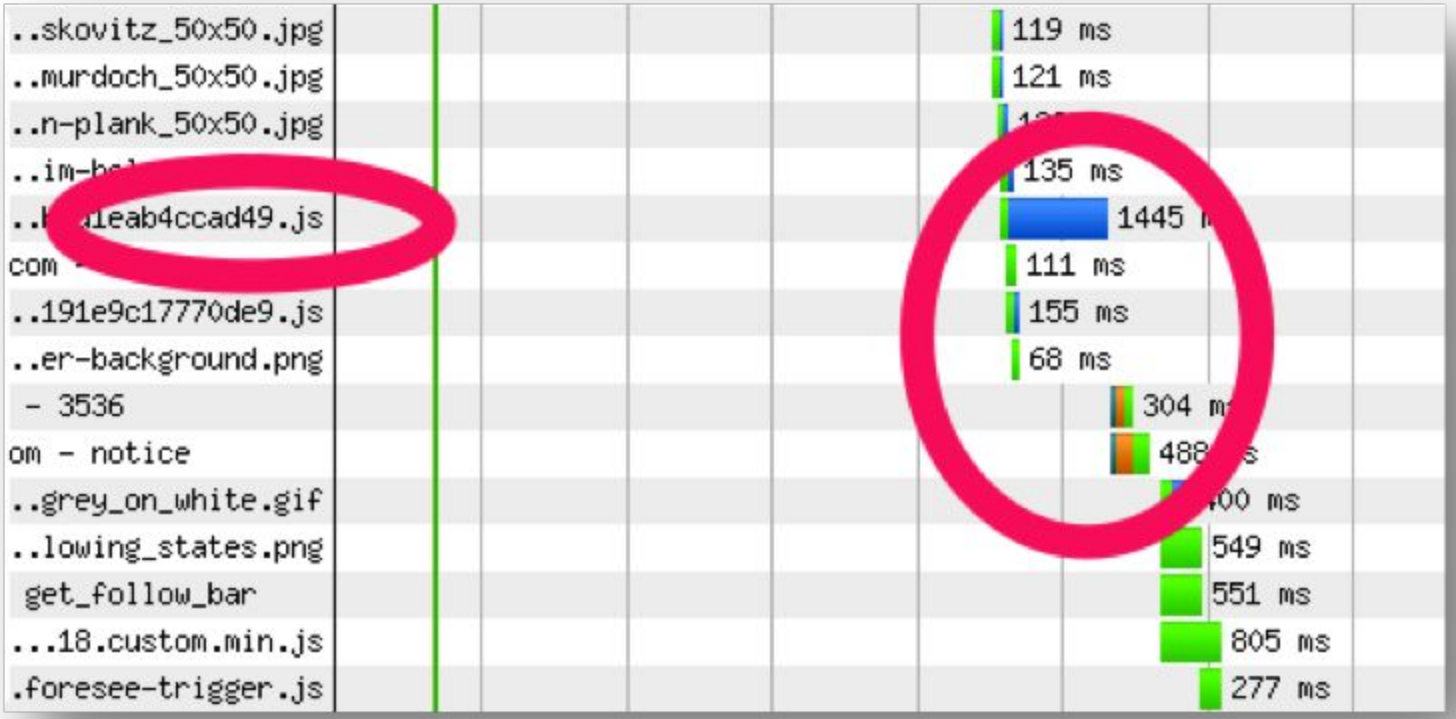
Motivation

- "*If the web is the human body, Javascript tags are like its nerve endings*" - www.krux.com
  - Means through which pages sense, respond, execute, measure, and remember
- Problem
  - Tag activity slows down pages
    - 0.1s delay can translate into a 1% drop in customer activity (Amazon)
    - 30% variance in page views based on load time (AOL)
    - Load times are increasingly important in Google and Bing search rankings

Javascript is synchronous

- Javascript is the #1 cause of slow web pages
  - Blocking behavior
  - When it loads nothing else happens
- Browsers render page elements synchronously
  - One element cannot load until the one before it has
- Solution
  - Load elements in a non blocking manner
    - Asynchronously

Example



http://www.krux.com/blog/krux-engineers/synchronous-versus-asynchronous-tags-whats-the-big-deal/

Why is JS not asynchronous?

- *document.write* construct
  - Inserts something into a web page
    - Text, tag, etc.
  - It expects to alter the page inline, inserting content as the page is loaded
    - Messes the content if loaded asynchronously
- Why do we still use it?
  - We cannot get rid of it until the entire stack can guarantee that *document.write* will not be used

Solutions

- **Element creation**
  - document.createElement(‘div’)
- **InnerHTML**
  - element.innerHTML(‘some content’)
- **Libraries** (JQuery library)
- **Iframes**
  - Work only if access to top level page is not needed
- **HTML 5 attributes** for the *script* element
  - Only work if the script does not use document.write

Load scripts asynchronously

- Do not wait for scripts to render page
- Solution
  - New HTML 5 *script* attributes
    - defer: script loads after page finished loading
    - async: script loads concurrently with the page
- Example

```
<script type="text/javascript" src="busy.js" async></script>
```

Example

- Javascript blocking code

```
var startNow = new Date();
var pauseFor = 5000; // In milliseconds
while (new Date() - startNow < pauseFor) ;
```

- HTML code

```
<html>
    <head>
        <script type="text/javascript" src="busy.js"></script>
        <script type="text/javascript">alert("Time's up"); </script>
    </head>
</html>
```

Outcome

– The alert pops out **after** the 5,000ms timeout elapses

Example

- Javascript blocking code

```
var startNow = new Date();
var pauseFor = 5000; // In milliseconds
while (new Date() - startNow < pauseFor) ;
```

- HTML code

```
<html>
    <head>
        <script type="text/javascript" src="busy.js“ async></script>
        <script type="text/javascript">alert("Time's up"); </script>
    </head>
</html>
```

Outcome

– The alert pops out **before** the 5,000ms timeout elapses

Callbacks

- JS functions are first class objects
  - Their type is *object*
  - They can be stored in variables, passed as arguments, returned from functions, etc.
- Essence of callbacks
  - Pass a function as an argument to another function and later execute that passed-in function or even return it to be executed later
- Most widely used functional programming technique in JavaScript

Callbacks

- When passing a callback as argument we **do not call** the function

```
var data = getData( function (data) {
        alert("We have " + data)
});
```

- The function **will be called somewhere in the body** of the function

```
function getData(callback) {
        var data = … //read the data
        callback(data);
        return data;
}
```

Asynchronous data transfer

- Load data without blocking the interface
  - Facebook comments
  - Twitter tweets
  - Youtube movies
- Solution
  - Use **XMLHttpRequest** object
    - **Supported** by: Chrome, IE7+, Firefox, Safari, and Opera
    - **AJAX** (Asynchronous Javascript and XML)
      - XHTML + CSS + DOM + XML + XMLHttpRequest

Example

```
function loadXMLAsync (filePath) {
        var req = new XMLHttpRequest();
        req.open("GET", filePath, true);
        req.send(null);
        req.onreadystatechange = function () {
                if (req.readyState == 4) {
                        if (req.status == 200) {
                                xml = req.responseXML;
                                processXML (xml);
                        }
                        else
                                alert ("Error loading XML");
                }
        }
}
```

Notes

- For local access use *req.status == 0* instead of the HTTP code 200
- State: 0 – uninitialized, 1 – loading, 2 – loaded, 3 – interactive, 4 – complete

Processing incoming XMLs

- **DOM**
  - createElement
  - appendChild

```
function processXML(xml) {
        var imgs = xml.getElementsByTagName('image');
        body = document.getElementsByTagName("body");
        for (var i=0; i<imgs.length; i++) {
                var image = document.createElement("img");
                image.src = imgs[i].firstChild.nodeValue; //set src attribute
                body[0].appendChild(image);
        }
}
```

Older IE browsers

- < IE7
- Use *ActiveXObject* instead

```
if (window.XMLHttpRequest) {
 req = new XMLHttpRequest();
 req.overrideMimeType('text/xml'); }
else if (window.ActiveXObject) { //try to get the most modern implementation
 var list = ["Microsoft.XmlHttp","MSXML2.XmlHttp","MSXML2.XmlHttp.3.0","MSXML2.
XmlHttp.4.0","MSXML2.XmlHttp.5.0"];
 var ok = false;
 var i = 5;
 while (i >= 0 && ok == false) {
        try {
               req = new ActiveXObject(list[i]);
               ok = true;
        } catch (e) {}
        i--;
 }
}
```

Async vs. sync API

Rule of thumb

- If API
  - requires IO, or
  - heavy processing

  (>15ms) expose it asynchronously from the start

Building asynchronous APIs

- Design APIs to be asynchronous from the start

```
var data = getData()
alert("We have " + data);
```

  - **Freezes** the user interface until data is fetched

```
getData( function (data) {
    alert("We have " + data)
});
```

  - **Designed asynchronously** even if the app needs to be async or not in a later stage
  - Use **callbacks**

What’s next?

- JQUERY
- Server side programming
- Web services
- Cloud computing