Who's this guy?

Principal Front End Engineer

Contributor, Creator of YUI Test

Author

Lead Author

Contributor

Lead Author
@slicknet

(Complaints: @codepo8)
Does JavaScript performance matter?
After all, all browsers now have optimizing JavaScript engines

Tracemonkey/JaegerMonkey (3.5+)
V8 (all)
Squirrelfish (4+)
Chakra (9+)
Karakan (10.5+)
So our scripts are getting really, really fast
Old computers ran slow applications
Small amounts of CPU power and memory
New computers are generally faster but slow applications still exist.

More CPU + more memory = less disciplined application development
It's still possible to write slow JavaScript on the new, faster JavaScript engines
JavaScript performance directly affects user experience
is getting tired of javascript. All it does is slow down page navigation and add complicated layouts and consume zillion resources
Where to start?
The UI Thread
The brains of the operation
The browser UI thread is responsible for both **UI updates** and **JavaScript execution**.
Only one can happen at a time.
Jobs for UI updates and JavaScript execution are added to a **UI queue** if the UI thread is busy. Each job must wait in line for its turn to execute.
<button id="btn" style="font-size: 30px; padding: 0.5em 1em">Click Me</button>

<script type="text/javascript">
window.onload = function (){
    document.getElementById("btn").onclick = function(){
        //do something
    }
};

</script>
Before Click

UI Thread

UI Queue

time
When Clicked

UI Thread

time

UI Queue

UI Update

onclick

UI Update
When Clicked

UI Thread

UI Update

time

Draw down state

Click Me

UI Queue

onclick

UI Update
When Clicked

UI Thread

UI Update  onclick

UI Queue

UI Update
When Clicked

UI Thread

UI Update → onclick → UI Update

time

Draw up state

Click Me

UI Queue
No UI updates while JavaScript is executing
JavaScript May Cause UI Update

<button id="btn" style="font-size: 30px; padding: 0.5em 1em">Click Me</button>

<script type="text/javascript">
window.onload = function(){
    document.getElementById("btn").onclick = function(){
        var div = document.createElement("div");
        div.className = "tip";
        div.innerHTML = "You clicked me!";
        document.body.appendChild(div);
    }
};
</script>
A UI update must use the latest info available
Long-running JavaScript = Unresponsive UI
Responsive UI

UI Thread

UI Update  JavaScript  UI Update

time
Unresponsive UI

UI Thread

UI Update  JavaScript  UI Update

time
The longer JavaScript runs, the worse the user experience
The runaway script timer prevents JavaScript from running for too long.
Each browser imposes its own limit (except Opera).
Stop running this script?

A script on this page is causing Internet Explorer to run slowly. If it continues to run, your computer may become unresponsive.
Warning: Unresponsive script

A script on this page may be busy, or it may have stopped responding. You can stop the script now, open the script in the debugger, or let the script continue.


Don't ask me again

Stop script  Debug script  Continue
Safari

Slow Script

A script on the page file:///C:/Documents%20and%20Settings/Nicholas/Desktop/LongRunningScriptTest.htm is making Safari unresponsive. Do you want to continue running the script, or stop it?

Stop  Continue
Chrome

The following page(s) have become unresponsive. You can wait for them to become responsive or kill them.

LongRunningScriptTest.htm

Kill pages  Wait
Runaway Script Timer Limits

• Internet Explorer: 5 million statements
• Firefox: 10 seconds
• Safari: 5 seconds
• Chrome: Unknown, hooks into normal crash control mechanism
• Opera: none
Does JIT compiling help?
Interpreted JavaScript
JITed JavaScript (1\textsuperscript{st} Run)

UI Thread

<table>
<thead>
<tr>
<th>Compile</th>
<th>Execute</th>
</tr>
</thead>
</table>

time
JITed JavaScript (After 1st Run)

UI Thread

Execute

time
How Long Is Too Long?

“0.1 second [100ms] is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result.”

- Jakob Nielsen
Translation:
No single JavaScript job should execute for more than 100ms to ensure a responsive UI
Recommendation:
Limit JavaScript execution to no more than 50ms

measured on IE6 :)
Doing so makes your program awesome
Loadtime Techniques
Don't let JavaScript interfere with page load performance
During page load, JavaScript takes more time on the UI thread
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>
<body>
    <p>Hello world!</p>
    <script src="foo.js"></script>
    <p>See ya!</p>
</body>
</html>
Result

UI Thread

UI Update  JavaScript  UI Update

time
Result

UI Thread

Hello world!  foo.js  See ya!

time

Demo!
The UI thread needs to wait for the script to download, parse, and run before continuing.
Result

UI Thread

Hello world!  Download  Parse  Run  See ya!

Download time takes the longest and is variable
Translation:
The page doesn't render while JavaScript is downloading, parsing, or executing during page load
<!doctype html>
<html>
<head>
  <title>Example</title>
</head>
<body>
  <script src="foo.js"></script>
  <p>Hello world!</p>
  <script src="bar.js"></script>
  <p>See ya!</p>
  <script src="baz.js"></script>
  <p>Uh oh!</p>
</body>
</html>
The more scripts to download in between UI updates, the longer the page takes to render.
Technique #1: Put scripts at the bottom
Best Practices for Speeding Up Your Web Site

The Exceptional Performance team has identified a number of best practices for making web pages fast. The list includes 35 best practices divided into 7 categories.

1. Put Scripts at the Bottom
2. Make JavaScript and CSS External
3. Minify JavaScript and CSS
4. Remove Duplicate Scripts
5. Minimize DOM Access
6. Develop Smart Event Handlers

Put Scripts at the Bottom

tag: javascript

The problem caused by scripts is that they block parallel downloads. The HTTP/1.1 specification suggests that browsers download no more than two components in parallel per hostname. If you serve your images from multiple hostnames, you can get more than two downloads to occur in parallel. While a script is downloading, however, the browser won’t start any other downloads, even on different hostnames.

In some situations it’s not easy to move scripts to the bottom. If, for example, the script uses `document.write()` to insert part of the page’s content, it can’t be moved lower in the page. There might also be scoping issues. In many cases, there are ways to workaround these situations.

An alternative suggestion that often comes up is to use deferred scripts. The `defer` attribute indicates that the script does not contain `document.write()`, and is a clue to browsers that they can continue rendering. Unfortunately, Firefox doesn’t support the `defer` attribute. In Internet Explorer, the script may be deferred, but not as much as desired. If a script can be deferred, it can also be moved to the bottom of the page. That will make your web pages load faster.
<!doctype html>
<html>
<head>
   <title>Example</title>
</head>
<body>
   <p>Hello world!</p>
   <p>See ya!</p>
   <script src="foo.js"></script>
</body>
</html>
Put Scripts at Bottom

Even if there are multiple scripts, the page renders quickly
Technique #2: Combine JavaScript files
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>
<body>
    <p>Hello world!</p>
    <p>See ya!</p>
    <script src="foo.js"></script>
    <script src="bar.js"></script>
    <script src="baz.js"></script>
</body>
</html>
Each script has overhead of downloading
Combining all of the files limits the network overhead and gets scripts onto the page faster
<!doctype html>
<html>
<head>
<title>Example</title>
</head>
<body>
<p>Hello world!</p>
<p>See ya!</p>
<script src="foo-and-bar-and-baz.js"></script>
</body>
</html>
Technique #3: Load scripts dynamically
Basic Technique

```javascript
var script = document.createElement("script"),
    body;
script.type = "text/javascript";
script.src = "foo.js";
body.appendChild(script, body.firstChild);
```

Dynamically loaded scripts are non-blocking
Downloads no longer block the UI thread
<!doctype html>
<html>
<head>
  <title>Example</title>
</head>
<body>
  <p>Hello world!</p>
  <script src="foo.js"></script>
  <p>See ya!</p>
</body>
</html>
Using HTML `<script>`

UI Thread

Hello world!  Download  Parse  Run  See ya!

time
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <p>Hello world!</p>
    <script>
      var script = document.createElement("script"),
          body = document.body;
      script.type = "text/javascript";
      script.src = "foo.js";
      body.insertBefore(script, body.firstChild);
    </script>
    <p>See ya!</p>
  </body>
</html>
Using Dynamic Scripts

Only code execution happens on the UI thread, which means less blocking of UI updates.
function loadScript(url, callback) {

    var script = document.createElement("script"),
        body = document.body;
    script.type = "text/javascript";

    if (script.readyState) {  //IE <= 8
        script.onreadystatechange = function() {
            if (script.readyState == "loaded" ||
                script.readyState == "complete") {
                script.readyState = "complete";
                script.onreadystatechange = null;
                callback();
            }
        };
    } else {  //Others
        script.onload = function() {
            callback();
        };
    }

    script.src = url;
    body.insertBefore(script, body.firstChild);
}
Usage

loadScript("foo.js", function(){
    alert("Loaded!");
});
Timing Note:
Script execution begins immediately after download and parse – timing of execution is not guaranteed
Using Dynamic Scripts

Depending on time to download and script size, execution may happen before next UI update.
Technique #4: Defer scripts
<!doctype html>
<html>
<head>
    <title>Example</title>
</head>
<body>
    
    <p>Hello world!</p>
    <script defer src="foo.js"></script>
    <p>See ya!</p>

    <!-- even more markup -->

</body>
</html>
Support for `<script defer>`
Deferred scripts begin to download immediately, but don't execute until all UI updates complete (DOMContentLoaded)
Using `<script defer>`

Similar to dynamic script nodes, but with a guarantee that execution will happen last.
Timing Note:
Although scripts always execute after UI updates complete, the order of multiple `<script defer>` scripts is not guaranteed across browsers.
Technique #5: Asynchronous scripts
<!doctype html>
<html>
<head>
  <title>Example</title>
</head>
<body>
  <p>Hello world!</p>
  <script async src="foo.js"></script>
  <p>See ya!</p>
<!-- even more markup -->
</body>
</html>
Support for `<script async>`

3.6 Soon Soon Soon
Asynchronous scripts behave a lot like dynamic scripts.
Using `<script async>`

Download begins immediately and execution is slotted in at first available spot
Note:
Order of execution is explicitly not preserved for asynchronous scripts
Runtime Techniques
Ways to ensure JavaScript doesn't run away
function processArray(items, process, callback){
    for (var i=0, len=items.length; i < len; i++)
    {
        process(items[i]);
    }
    callback();
}
Technique #1: Timers
JavaScript Timers

• Created using setTimeout()
• Schedules a new JavaScript execution job for some time in the future
• When the delay is up, the job is added to the UI queue
  – Note: This does not guarantee execution after the delay, just that the job is added to the UI queue and will be executed when appropriate
JavaScript Timers

• For complex processing, split up into timed functionality
• Use timers to delay some processing for later
function timedProcessArray(items, process, callback) {
    // create a clone of the original
    var todo = items.concat();
    setTimeout(function() {
        var start = +new Date();
        do {
            process(todo.shift());
        } while (todo.length > 0 &&
                  (+new Date() - start < 50));
        if (todo.length > 0) {
            setTimeout(arguments.callee, 25);
        } else {
            callback(items);
        }
    }, 25);
}
When Clicked

UI Thread

time

UI Queue

UI Update

onclick

UI Update
When Clicked

UI Thread

UI Update

time

UI Queue

onclick

UI Update
When Clicked

UI Thread

UI Update

onclick

time

UI Queue

UI Update
When Clicked

UI Thread

UI Update  onclick  UI Update

time

UI Queue
After 25ms

UI Thread

UI Update  onclick  UI Update

UI Queue
JavaScript
After 25ms

UI Thread

UI Update  onclick  UI Update  JavaScript

time

UI Queue
After Another 25ms

UI Thread

- UI Update
- onclick
- UI Update
- JavaScript

UI Queue

JavaScript
After Another 25ms

UI Thread

UI Update  onclick  UI Update  JavaScript  JavaScript

time

UI Queue
Technique #2: Web Workers
Abstract

This specification defines an API that allows Web application authors to spawn background workers running scripts in parallel to their main page. This allows for thread-like operation with message-passing as the coordination mechanism.
Web Workers

- Asynchronous JavaScript execution
- Execution happens in a separate process
  - Not on the UI thread = no UI delays
- Data-driven API
  - Data is serialized when sending data into or out of Worker
  - No access to DOM, BOM
  - Completely separate execution environment
//in page
var worker = new Worker("process.js");
worker.onmessage = function(event){
    useData(event.data);
};
worker.postMessage(values);

//in process.js
self.onmessage = function(event){
    var items = event.data;
    for (var i=0,len=items.length; i < len; i++){
        process(items[i]);
    }
    self.postMessage(items);
};
When Clicked

UI Thread

UI Queue

- UI Update
- onclick
- UI Update

time
When Clicked

UI Thread

UI Update

time

UI Queue

onclick

UI Update
When Clicked

UI Thread

UI Update  onclick

UI Queue

UI Update
When Clicked

UI Thread

UI Update

onclick

Worker Thread

time

UI Queue

UI Update
When Clicked

UI Thread

UI Update  onclick  UI Update

Worker Thread

JavaScript

UI Queue
Worker Thread Complete

UI Thread

UI Update
onclick
UI Update

time

UI Queue
onmessage
Worker Thread Complete

UI Thread

- UI Update
- onclick
- UI Update
- onmessage

UI Queue

time
Support for Web Workers

3.5  4.0  4.0  ?  10.6
Recap
The browser UI thread is responsible for both **UI updates** and **JavaScript execution**. Only one can happen at a time.
Responsive UI

UI Thread

UI Update  JavaScript  UI Update

time
Unresponsive UI

UI Thread

UI Update  JavaScript  UI Update

time
Avoid Slow Loading JavaScript

• Put scripts at the bottom
• Concatenate scripts into as few files as possible
• Choose the right way to load your scripts
  – Dynamically created scripts
  – Deferred scripts
  – Asynchronous scripts
Avoid Slow JavaScript

• Don't allow JavaScript to execute for more than 50ms
• Break up long JavaScript processes using:
  – Timers
  – Web Workers
The End
Etcetera

• My blog:  
  www.nczonline.net

• Twitter:  
  @slicknet

• These Slides:  
  http://slideshare.net/nzakas/presentations/

• Rate Me:  
  http://spkr8.com/t/4568
Questions?
Creative Commons Images Used

- http://www.flickr.com/photos/8628950@N06/1332225362/
- http://www.flickr.com/photos/hippie/2406411610/
- http://www.flickr.com/photos/55733754@N00/3325000738/
- http://www.flickr.com/photos/eurleif/255241547/
- http://www.flickr.com/photos/off_the_wall/3444915939/
- http://www.flickr.com/photos/wwarby/3296379139/
- http://www.flickr.com/photos/derekgavey/4358797365/
- http://www.flickr.com/photos/mulad/286641998/