Evolution of Web App Architectures, Revisited

Goes with examples in javascript2 directory

Most of these notes courtesy of M. Hall coreservlets.com. Used by permission

The web browser as a runtime engine

Revisiting the evolution of the Web:

- Originally web servers served *documents*
- Then, the document was viewed as a collection of *content fragments*
- Meanwhile, some folks figured out and hacked together a way to have the document content dynamically generated (CGIs)
  - So we had dynamic behaviors with HTML output
- Then, little dynamic fragments were inserted into static content
  - This is where SSIs and server-side scripts came in (and have stayed)
- Then the web became an application-serving platform more than just a content platform (though content consumption remains a killer app)

As the web moves toward this dynamicism (is that a word?), viewing the browser as a runtime environment is important

- Javascript, which started as a simple way to make a content-driven web page more “active” has evolved into the dominant web language

Yes there is a standard, but variations b/w browsers remain

- It is called EcmaScript, we will come back to this later
  - http://www.ecma-international.org/publications/standards/Ecma-262.htm

The browser as a runtime engine

NOTE: Please review Tali Garsiel's paper!

The web (including mobile apps backed by web servers) is now primarily an app delivery platform for different apps:

- E-commerce, Content management and delivery
- Social media (participatory just-in-time content interaction)
- Educational delivery, healthcare delivery, financial services delivery
- *…different types (verticals) and different commonalities (horizontal)*

- Common functional aspects of web applications:
  - Rendered primarily (but not exclusively) through HTML
  - Still has remnants of "click-and-wait" (*...stay tuned*)
- Computationally:
  - The runtime environment of a browser consists of the browser's ability to support extensions, it's rendering engine, and it's JS engine

We will start by revisiting Javascript in the context of manipulating content

Continuing Evolution of Web Architecture

As the web became app-centric, developers realized:

- We can write full apps in Javascript
  - No server, no “page turns”, single-page apps
- Our apps can be more responsive to end user interaction
  - Reduce if not eliminate “click and wait”
- Moving back to “fat clients” has benefits, and drawbacks
  - We can use the inherent power of increasingly diverse devices in powerful and personalized ways (pro)
  - We really are getting back to a desktop-like “app-y” model (pro)
  - The architecture though is way different (pro or con?)
    - We alleviate computational burden on servers (pro)
    - We better enable the cloud and mobile computing (pro)
    - We (may be) less secure and less private (con)
    - It is harder to test (functionally and performance) in such an env (con)

Pros or cons, this is the way app development is moving

- We will discuss this trend, and how it affects the principles we have discussed so far for web development

Continuing Evolution of Web Architecture

So our updated web app architecture evolution looks like this:

1. CGIs
2. SSIs/server-side scripts
3. Implicit invocation (server-based handlers)
4. Components and Containers (servlet model)
5. Single-threaded asynchronous event queue (NodeJS)
6. Browser-centric applications
    1. Single-page applications
    2. Interactive applications with access to world model state (AJAX)
    3. Full-blown RESTfully API-driven applications

Full disclaimer: again yes this is an over-simplification of web app evolution, (reality is far messier/nonlinear) and I know I am leaving out lots of your friends' favorite web technologies. But I believe it is an acceptable framework for understanding what has happened for 20+ years!

Javascript: Loading Scripts on a Web Page

To include Javascript with your rendered HTML response, you either include it inline or by reference

script with content (inline)

- `<script type="text/javascript">`*`JavaScript code here`*`</script>`
  - Purpose
    - To directly invoke code that will run as page loads
      - E.g., to output HTML content built by JavaScript
    - Don't use this approach for defining functions or for doing things that could be done in external files.
      - Slower (no browser caching) and less reusable

script with src (by reference)

- `<script src="my-script.js" type="text/javascript”></script>`
  - Purpose
    - To define functions, objects, and variables.
    - Functions will later be triggered by buttons, other user events, inline script tags with body content, etc.

Example (*phish.js + loading-scripts.html*)

```
function getMessage() {
  var amount = Math.round(Math.random() * 100000);
  var message ="You won $" + amount + "!\n" +
    "To collect your winnings, send your credit card\n" +
    "and bank details to oil-minister@phisher.com.";
  return(message);
}
function showWinnings() {
  alert(getMessage());
}
function showWinnings2() {
  document.write("<h1><blink>" + getMessage() + "</blink></h1>");
}
```

"alert" pops up dialog box

"document.write" inserts text into page at current location

```
<html><head><title>Loading Scripts</title> // stuff...
<script src="phish.js" type="text/javascript"></script>
</head>
<body> // more stuff...
<input type="button" value="How Much Did You Win?"
   onclick='showWinnings()'/>
<script type="text/javascript">showWinnings2()</script>
</body></html>
```

Loads script from above

Calls showWinnings1 when user presses button. Puts result in dialog box.

Calls showWinnings2 when page is loaded in browser. Puts result at this location in page.