Webpages that don't suck

Rolfe Bozier

3-Mar-2015

Agenda

- Quick intro to HTTP and HTML
- CSS – managing appearance
- CGI – dynamic content and user interaction
- Server-side scripting – app development
- JavaScript – client-side scripting
- Cookies – stateful web applications
- AJAX – asynchronous updates

Introduction to HTTP and HTML

- HTTP is the protocol used for communications between a web server and a browser
- HTTP is a stateless protocol
  - a browser sends a request to a web server for some resource
  - web server sends back content
  - web server forgets everything
- Most common resource is a page marked up in HTML
- but also:
  - Images
  - Flash objects
  - style sheets
  - javascript

Introduction to HTTP and HTML

- HTML defines the structure of page content
- Markup includes:
  - headings, paragraphs, fonts, colours
  - images, imagemaps, links to other pages
  - lists
  - forms (buttons, textboxes, dropdowns)
  - tables
- HTML pages are downloaded and rendered in your browser
- Some advantages of HTML pages:
  - structured output
  - formatted output (default format provided by browser)
  - low transmission overhead
  - everyone has a browser

- [Example 1]

Controlling appearance using CSS

- The browser's default rendering is pretty boring
- Advantages in customising the appearance
  - page looks nicer – web publishing becomes a thing
  - make the content easier to parse
  - web sites can have distinctive style / branding
- A brief aside: the DOM
  - browser creates a tree representation of the page from the HTML markup
  - this is the Document Object Model
  - it is a cross-platform convention
  - the nodes ("elements") and tree structure are defined by the HTML
  - each element has additional appearance ("style") attributes
- Controlling the appearance of a page is done with Cascading Style Sheets ("CSS")
  - CSS is another language
  - It defines the appearance and layout of an element in the DOM
  - This can be inherited by lower elements in the DOM
  - CSS lets you override just about anything for an element

Controlling appearance using CSS

- You can provide CSS information by:
  - including it in an HTML page using <style>...</style> tags
  - referencing an external style sheet: <link rel="stylesheet" ... /> tag
- Style information is defined separately from the HTML content
- … so you need a way to reference HTML elements:
  - by element type, e.g. table, body, p
  - by element "class" - multiple elements in a class, element can have multiuple classes
  - by element "id" - unique identifier for one element
  - relative to one of the above, e.g. all tables in a "fancy" div

- [Example 2]

- So, now we can provide static information with structure and appearance

Dynamic content

- Run a *program* on the server that *generates* the HTML content
  - this is a CGI program
  - It can be any executable program
  - Program's output is HTML data that is sent back to the browser by the web server
  - CGI program are general programs so they can do anything:
    - retrieve data from a database
    - summarise content from local data files
    - retrieve data from hardware (e.g. webcam)

User interaction

- CGI program can also be driven by input from the user
  - classic example: HTML form submitting
    - HTML form data sent to CGI program
    - …which then sends back "the results"
  - or you can bypass the form and just pass data as extra arguments to the URL - the query string
    - http://somehost/someurl?param1=1&param2=hello
- HTML forms have two ways of passing data to the CGI program:
  - GET: pass form values at the end of the URL
  - POST: include in HTTP request not as part of the URL
- GET normally used for queries
- POST used for state-changing operations

- [Example 3]

Dynamic content and user interaction

- But still this is stateless
  - all information shared between requests is stored in the returned HTML page
- Also, CGI programs which accept user input can be a security issue
  - CGI executes on the web server
  - needs to absolutely protect against broken or malicious input
  - this can be harder than you think

Dynamic content via server-side scripting

- Another way to generate content is to use server-side scripts
- Web pages are really scripts in some language
  - the script looks like a normal URL
  - the script is executed on the server
  - execution results are merged with static content to create HTML output
  - the HTML output is sent to the browser
  - somewhere in between static HTML and a pure CGI executable
- Typical languages:
  - PHP
  - ASP
  - ColdFusion
- Most common scripting languages using an HTML-generation and/or templating modules

- [Example 4]

Client-side scripting - JavaScript

- We can also run scripts in the browser using JavaScript
- What can we do with this?
  - add event handlers to the DOM
    - trigger on page load
    - trigger on some user interaction (e.g. button, mouseover)
  - we can change the DOM (structure and/or style)
    - show or hide elements
    - adjust style based on some local information
    - validate user input before it is sent
    - create interactive widgets

Client-side scripting - JavaScript

- Advantages
  - improve the utility of web pages
  - save expense of round-trip to server for simple actions
- How to add it
  - including it in an HTML page or reference an external script using <script> tags
  - the sky is the limit with what you can do
  - rather than creating lots of new JavaScript, look for existing toolkits
    - jQuery is one example
- [Example 5, 6]

Breaking the stateless shackles - cookies

- A cookie is a blob of state data the web server keeps in your browser
  - associated with a specific URL
  - web server includes cookie with HTTP reply
  - browser sends cookie if there is one for the domain
- Now HTTP is not stateless any more
  - a session is multiple HTTP requests/replies linked by a cookie
- Pretty much essential for banking, ecommerce etc
  - web server needs to keep track of users for secure interactions
  - cookie contents need to be protected for web server security
  - malicious users can't be able to fake someone else's cookie
- Cookies can also be used for other purposes, such as behavioural tracking or preference storage
- You probably won't care about cookies, but sometimes they can be handy

- [Example 7]

Breaking the synchronous shackles - AJAX

- Up to now, all interactions between browser and server are synchronous
  - nothing major happens until you submit a new HTTP request e.g. click on a form
  - but this is an expensive operations - the entire page is discarded and a new one sent from the server
  - plus it looks ugly seeing the page redrawn
- There is/was a facility called *server push*
  - web server keeps sending updated HTML replies back to the browser
  - ugly and unscalable, no-one in their right mind uses this any more
- AJAX is a JavaScript facility for sending *asynchronous HTTP requests* to the server
  - Initiated by some event on the client side
  - Send a request via HTTP for some data (may not be HTML)
  - Asynchronous action request, need a javascript callback to handle the reply
  - The callback can do things like update the DOM
  - A common format for AJAX data is JSON

- [Example 8]

The future (actually now)

- HTML5 was recently standardised (HTML4 is 18 years old)
  - <canvas>, <video>, <audio> tags
  - More semantic markup tags
  - SVG integration
  - DOM support now standardised
  - new APIs for DOM interaction e.g. drag-and-drop
  - deprecation of appearance-only tags (should be done in CSS)
  - support for DRM ☹
- Browsers already support HTML5

Web applications

- Fully-blown web applications are another level again
  - HTTP shrinks to a tiny part
  - Browser runs a UI based on a JavaScript framework
    - Angular.js, Io.js, Ember.js, Backbone.js, etc.
  - Extended HTML is parsed in the browser
  - AJAX becomes the default
  - Data exchange using SOAP, XML, JSON, …

Summary

- Web pages are an excellent mechanism for deploying content to users
  - Everyone has a browser
  - These days you generally don't need to worry about varying support between browsers
- There is a continuum of complexity from simple HTML+CSS to full-blown interactive browser-based applications
- There are always opportunities and reasons to deploy HTTP-based tools
  - you can easily slot in at the complexity level you need
  - maybe later you can add new features
  - there are infinite resources on the Internet to learn this (only a slight exaggeration)