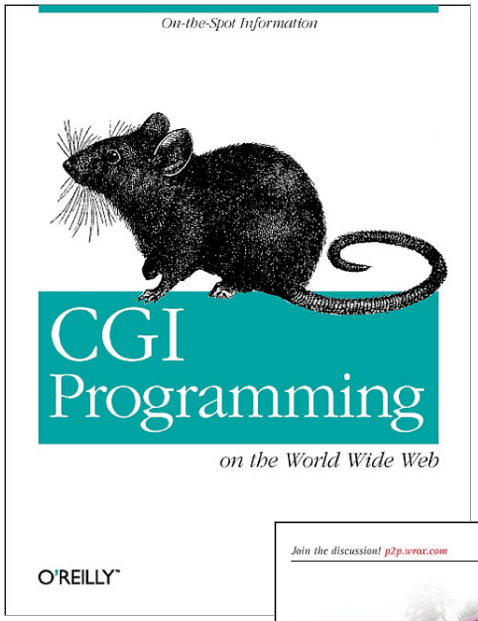
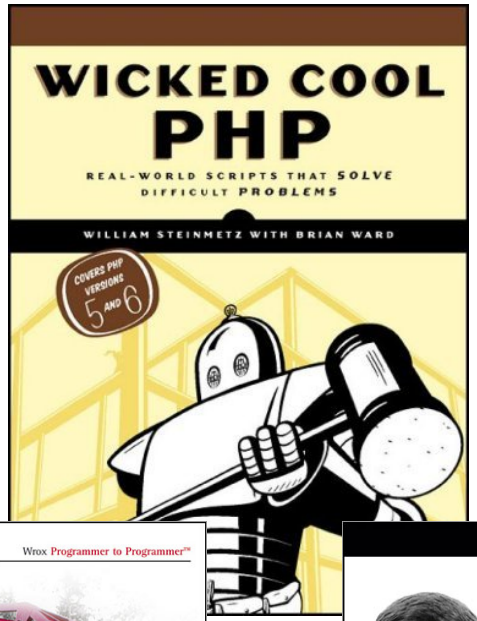
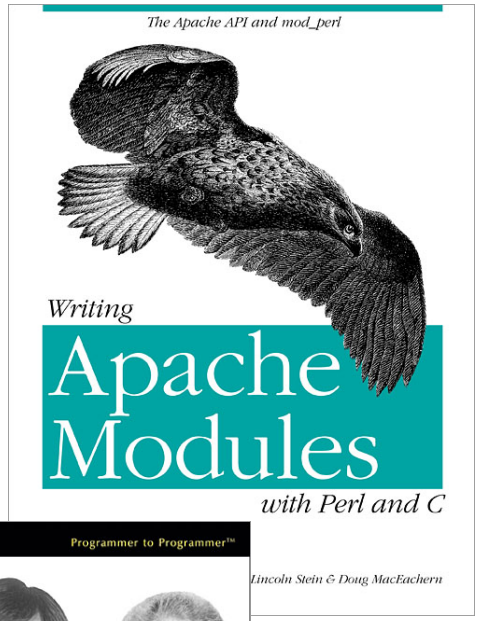
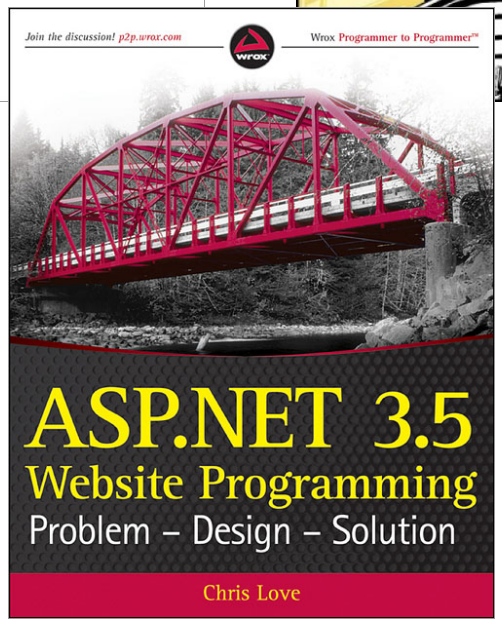
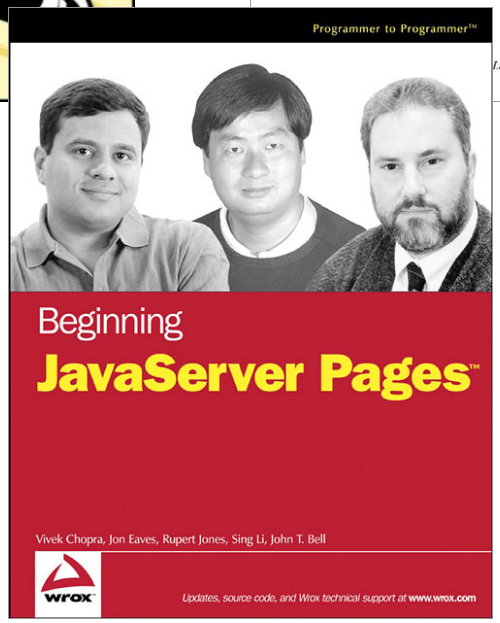
Dynamic web content technologies











Overview

- Dynamic content
  - What it is
  - Sources of input
- CGI (Common Gateway Interface)
  - FastCGI
- Server-side scripting
  - PHP, ASP, JSP
- Web server modules
- Custom web server

Static vs. dynamic

- Static content
  - Images and pages don't change
    - Always the same, like a file server
  - Fast to deliver, easy to cache
- Dynamic content
  - Same URL results in different delivered HTML
    - e.g. different preference on # of products to display
  - May change as user interaction progresses
    - e.g. adding items to a shopping cart
  - Need something besides just HTTP and HTML
    - HTTP is stateless
    - HTML is not programmable (e.g. conditional, loops)

Input to dynamic pages

- Form fields
  - <INPUT> tags inside the <FORM> tag
  - URL encoding (percent-encoded)
    - if GET: in the URL, if POST: in the HTTP payload
    - Unreserved characters:
      - ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz0123456789-_.~
    - Reserved characters:
      - !*'();:@&=$,/?#[]
      - Converted to %XX, where XX is ASCII in hexadecimal
      - %20 = space (also +), %21 = !, %23 = #, %25 = %, …
    - Most languages have URL encode/decode functions

Input to dynamic pages

- Cookies
  - Differentiate clients hitting same page
- Other input sources:
  - User agent (browser)
  - HTTP referer
    - Misspelled since original RFC 1945
    - The page you came from to get to this one
  - Client's IP address
  - Time of day
  - …

CGI

- CGI (Common Gateway Interface)
  - In use since 1993
  - Requests a URL in a special location/file extension
    - e.g. http://www.blah.com/cgi-bin/lookup
  - Web server passes request to script/program
    - Sets a laundry list of environment variables
    - Creates new process and runs program
    - Program's output sent to web client
  - Notes:
    - Program needs read + execute permissions
      - By whatever user web server running under
      - Probably shouldn't be world writeable

CGI pros

- Advantages
  - Simple to code
    - HTTP GET: input from environment variables
    - HTTP POST: input from standard input
    - Output via standard output
  - Highly portable
    - Supported by virtually all web servers
    - Especially if written in portable scripting language
  - Safer?
    - Runs outside web server process space
      - Can run as different user
    - But subject to exploits if you're not careful

CGI cons

- Disadvantages: expensive
  - Takes time
    - Most fork process for each HTTP request
    - Process has to start resources
      - e.g. Python interpreter, DB connection
  - Takes memory
    - Each process consumes resources
    - Particularly for interpreted languages

How expensive is CGI?

- Creating/destroying a process?
- Starting/ending Perl?
- Starting/ending Python?
- Experiment:
  - Test options on a page that doesn't do anything
  - Isolates process startup expense
  - Isolates interpreter startup expense

```
#!/usr/bin/perl

print "Content-Type: text/plain;charset=us-ascii\n\n";
print "noop!\n";
```

CGI noop results

- Test setup:
  - Pentium 4 @ 3.0 Ghz, 1GB memory, 512K L2 cache
  - Apache 2.2.21
  - Apache benchmark (ab) to localhost noop pages
    - 1000 requests each at concurrency {1, 5, 10, 20}
    - Variants: No CGI, CGI C, CGI Perl, CGI Python

Test case	Requests / second
No CGI	1272.2
CGI C program	389.7
CGI Perl script	236.1
CGI Python script	24.7

Improving CGI

- FastCGI
  - Developed by third party: Open Market
  - Language independent (like CGI)
    - Code in: C, C++, Java, Python, Perl, Ruby, …
  - Isolated from web server process space (like CGI)
  - Supported by many servers
    - e.g. Apache, IIS, Lighttpd, Nginx, …
- Benefits:
  - Faster! Persistent process serves many requests
    - Amortizes process/interpreter startup
  - Can proxy work to another server

FastCGI sample code

```
#!/usr/bin/perl
# FastCGI example in Python

use FCGI;
use strict;

my $i = 0;
my $request = FCGI::Request();

while ($request->Accept() >= 0)
{
    print "Content-Type: text/plain;charset=us-ascii\n\n";
    print "Hello world! i = " . $i;
    $i++;
}
```

```
// FastCGI example in C
#include "fcgi_stdio.h"
#include <stdlib.h>

int main(void)
{
    int count = 0;
    while (FCGI_Accept() >= 0)
    {
        printf("Content-Type: text/plain;charset=us-ascii\n\n");
        printf("Hello world! count = %d\n", count);
        count++;
    }
}
```

Server-side scripting

- Server-side scripting
  - Do smart stuff inside web server process
    - No process/interpreter startup expense
    - But still interpreted
  - Embed code into your HTML page
    - `<?php echo "hello world!"; ?>`
    - Results of running code sent to client, not code itself
  - Many language choices:
    - PHP: Hypertext Preprocessor
    - ASP (Active Server Pages), now ASP.NET
    - JSP (Java Server Pages)
    - ColdFusion

PHP scripting

- PHP: Hypertext Preprocessor
  - For documentation see http://php.net
  - Syntax: C/Java/Perl like
  - Variables
    - Prefix with $
    - Loosely typed
    - Don't need to declare before use
  - Hundreds of base functions:
    - e.g. File I/O functions:
      - fopen(), fgets(), fputs(), fclose(), feof()
  - Normally interpreted
    - But some compiled, e.g. HipHop by Facebook

PHP script: server-side

```
<html>
  <head>
    <script language="JavaScript">
       var secret = Math.floor(Math.random() * 10) + 1;
    </script>
  </head>
  <body onload="document.getElementById('num').innerHTML = secret;">
    Pssst, the secret number is <span class="num" id="num"></span>
    <br /><br />
    Your IP address is :
    <?php echo $_SERVER["REMOTE_ADDR"]; ?>
    <br /><br />
    <?php
      for ($i = 0; $i < 10; $i++)
         echo $i . " ";
      echo "<br />";
    ?>
  </body>
</html>
```

As seen on the web server.

Content deliver to client

```
<html>
  <head>
    <script language="JavaScript">
       var secret = Math.floor(Math.random() * 10) + 1;
    </script>
  </head>
  <body onload="document.getElementById('num').innerHTML = secret;">
    Pssst, the secret number is <span class="num" id="num"></span> <br />
    <br />
    Your IP address is 150.131.139.232 <br />
    <br />
    0 1 2 3 4 5 6 7 8 9 <br />
  </body>
</html>
```

As seen in client's browser.

Web server module

- Put interpreter into web server process space
  - e.g. mod_perl, mod_python
  - No startup expense

```
PerlModule ModPerl::Registry
  <Location /perl>
    SetHandler perl-script
    PerlHandler ModPerl::Registry
    Options ExecCGI
    allow from all
    PerlSendHeader On
  </Location>
```

Going even faster…

- Server-side scripting or mod_perl/python
  - Normally interpreted
  - Not as fast as native code
- Put native code in web server process
  - e.g. Apache module, IIS ISAPI, Netscape NSAPI
  - Fast!
    - No startup expense
    - Runs as native compiled code
  - But:
    - Not as portable, tied to specific web server
    - Web server process exposed to bugs in module

Apache hello world module

```c
#include "httpd.h"
#include "http_config.h"
#include "http_protocol.h"
#include "ap_config.h"

/* The sample content handler */
static int poc_rest_handler(request_rec *r)
{
    if (strcmp(r->handler, "poc_rest"))
        return DECLINED;
    r->content_type = "text/html";
    if (!r->header_only)
        ap_rputs("Hello world!\n", r);
    return OK;
}

static void poc_rest_register_hooks(apr_pool_t *p)
{
    ap_hook_handler(poc_rest_handler, NULL, NULL, APR_HOOK_MIDDLE);
}

/* Dispatch list for API hooks */
module AP_MODULE_DECLARE_DATA poc_rest
    STANDARD20_MODULE_STUFF,
    NULL,                  /* create p
    NULL,                  /* merge  p
    NULL,                  /* create per-server config structures */
    NULL,                  /* merge  per-server config structures */
    NULL,                  /* table of config file commands       */
    poc_rest_register_hooks  /* register hooks                      */
};
```

```
# My custom hello world Apache module
LoadModule hello_module modules/mod_hello.so
<Location /hello>
  SetHandler hello
</Location>
```

Even faster?

- What could be faster than native code inside web server process?
  - Create custom web server
    - HTTP is a simple protocol
    - Why reinvent your own socket protocol?
    - Only handles HTTP requests for specific app
  - Could be deployed on separate servers
  - Perhaps faster
    - No overhead associated with lots and lots of features
  - Smaller memory footprint
    - Important for scalability

Summary

- Dynamic content
  - Many ways to skin a cat?
  - Each has pros and cons
- CGI (Common Gateway Interface)
- FastCGI
- Server-side scripting
- Web server modules
- Building a custom web server