Creating Dynamic Websites with CGI and Mason - Day Two

Jon Warbrick
University of Cambridge Computing Service

Introducing Mason

What's wrong with CGI?

- Mixing code and HTML is a bad idea
- Repeated re-execution means
  - CGI has a large execution overhead
  - No persistence
- No access to webserver API

The Mason solution

- Mason is a 'Perl-based web application environment'
- Uses 'templates' to generate HTML
- Uses mod_perl
  - mod_perl embeds a Perl interpreter into Apache
  - also provides access to the Apache API
- Needs some tedious installation/configuration - we'll assume this has been done
- What follows assumes a 'CS standard' Mason installation

A simple Mason document

- Example 20: *mason.html*:

```
<html>

<head>

<title>A first Mason document</title>
</head>

<body>
<h1>Hello World</h1>
<p>Here we all are again</p>
</body>

</html>
```

A slightly more interesting Mason program

- Example 21: *date.html*

```
% my $now = localtime();

<html>

<head>
<title>A second Mason document</title>
</head>

<body>
<h1>Hello World</h1>
<p>It is <% $now %></p>
</body>

</html>
```

Mason from 10,000 feet

Components

- A combination of HTML and Mason markup
- Default is HTML
- HTML is output verbatim
- Mason markup contains Perl and Mason directives
- A component can represent
  - a page intended to be served directly - a 'top level component'
  - or part of a page (hence 'component')
- We have 'libraries' of components to do things like add the University House Style

Component syntax - embedded Perl

- A line starting `%` is interpreted as Perl code

```
% my $now = localtime();
```

- Best used to impliment Perl flow control structures

```
% if ($day eq 'Friday') {
<p>Going home early</p>
% }

<ul>
% foreach (1..3) {
<li>Here we go!</li>
% }
</ul>
```

Component syntax - Perl blocks

- Lines enclosed between `<%perl>` and `</%perl>` are interpreted a blocks of Perl code for execution
- Equivalent to, though probably better than, multiple lines starting `%`

```
<%perl>
my $who = 'Fred Smith';
my $date = localtime();
</%perl>
```

- Perl code in a `<%init>` block is equivalent to a `<%perl>` block at the start of the component
  - But it can appear anywhere
  - Convenient for 'hiding' Perl code needed to setup things for the rest of the component

Component syntax - substitution

- Anything between `<%` and `%>` tags is evaluated and substituted
- Typically used to substitute variables defined elsewhere

```
<p>Welcome, <% $who %>, it's now <% $date %></p>
```

- Values can (and generally should) be HTML-escaped by adding `|h` before the closing tag

```
<p>Welcome, <% $who |h %>, it's now <% $date |h %></p>
```

- `|u` requests URL escaping
- A default, typically `|h`, can be set
  - in which case `|n` request no escaping

Component syntax - calling other components

- Something like `<& header.mason &>` is replaced by the result of a call to the component *header.mason*
- Component names can be:
  - relative to the current component
  - relative to the *component root*, typically *document root*
  - extracted from perl expressions (but beware of some magic)
- Component libraries - see Example 22: *hs-mason.html*

Component syntax - other things

- Comments can appear
  - on lines starting `%#`
  - or within `<%doc>` and `</%doc>` blocks

```
%# This is a comment
<%doc>
As is all of this
...and this
...and this
</%doc>
```

- There are some other `<%...>` and `</%...>` blocks - we'll come across some later
- If a line ends `\` (backslash) then the backslash and the following newline are ignored
- Two special global variables let you interact with Mason and Apache
  - `$m` - the 'Mason object'
  - `$r` - the Apache request object

Passing information to components

Calling components

- All components can be called with arguments
- For a component invoked by a HTTP request, argument names and values come from the request:
  - 'query string' for GET requests
  - the request body for POST requests
- Otherwise arguments are supplied in the call
- There are (at least) two ways for a component to access its arguments:
  - via an `<%args>` block
  - via the `%ARGS` variable

Arguments via a `<%args>` block

- A component can declare the names and types of the arguments it expects in a `<%args>` block
- Types are declared by the initial character
  - `$` for a simple 'scalar' variable
  - `@` for a list 'array'
  - `%` for a lookup 'hash'
- The block can optionally include default values
- Arguments with no default are required
- Argument values are available from identically-named variables

```
<%args>
$name               # user's name
@dates              # a list of dates
# the rest are optional
$age => 21
@values = (9, 21, 432)
</%args>
```

Arguments via `%ARGS`

- A Perl 'hash' called **`%ARGS`** contains all of the arguments with which the component was called
- Necessary if parameter name can't be Perl variables
- The hash keys are the argument names
- The corresponding values contain the arguments
- Arrays and hashes are passed as references

Argument passing examples

- Consider a component with a **`<%args>`** like this

```
<%args>
$name
@colour
</%args>
```

- It could be called with a query string like this

```
example.html?name=John%20Smith&colour=red&colour=blue
```

- or from another Mason component like this

```
<& example.mason, name => 'John Smith',
                  colour => ['red', 'blue'] &>
```

- If **`@colour`** was **`$colour`** it would recieve a *reference* to the list of colours
- In both cases
  - **`%ARGS`** would be **`( name => 'John Smith', colour => ['red', 'blue'] )`**

Autohandlers and Dhandlers

Automatic content wrapping

- It's common to want standard headers and footers, navigation bars, etc
- Doing this by hand is tedious and hard to maintain
- When processing a component, Mason looks for a component called autohandler.mason in the same directory
- If it can't find one it looks in the next directory up, and so on
- At the point where it wants to insert the original component, the autohandler should call **`$m->call_next`**
- Example 23: *autohandler.mason, wrap1.html, wrap2.html*

Providing default content

- If asked for a component that doesn't exist
  - Mason first looks for a component called dhandler.mason in the same directory as the missing component
  - If it doesn't find it it looks in all parent directories
- If it finds a dhandler it processes that instead of the requested component
- ...and makes the rest of the component path available by calling **`$m->dhandler_arg`**
- The dhandler can then generate what content it likes
- Example 24: *dhandler.mason*

Doing 'CGI' things in Mason

Forms

- Forms are fairly straight forward - see Example 25: viewer2.html
- The only problem is arranging for 'sticky' fields
- One approach is to use **`cgi.mason`** - see Example 26: viewer3.html

Getting information about the request

- For CGI environment variable information, use the Apache request object. For example
  - Request method: **`$r->method()`**
  - Remote user: **`$r->connection->user()`**
  - ... or **`$r->user()`** (Apache 2)
  - User-agent header: **`$r->headers_in()->{'User-agent'}`**
- Most (all?) CGI environment variables also available
- Example 27: *info1.html*

Sending response meta-information

- No need (or support) for the 'special' CGI headers
- Content type normally defaults correctly based on filename
  - `$r->content_type('text/html; charset=utf-8')`
  - Example 28: *text.html*
- Redirect
  - `$m->redirect($new_url)`
  - Example 29: *random3.html*
- Return with a non-200 status (e.g. 'Not found')
  - `$m->clear_buffer;`
  - `$m->abort(404);`
  - Example 30: *forbidden.html*
- Setting other response headers
  - `$r->headers_out->{'X-panic'} = 'Now!'`
  - Example 31: *panic.html*

Debugging Mason

- Syntax and run-time errors reported
  - to the browser (in development)
  - to the Apache error log (in production)
  - messages can be confusing, line numbers can be wrong
  - Example 32: *syntax.html*, *runtime.html*, *confusion.html*
- Write your own log messages with

```
$r->log->emerg('A emergency!');
$r->log->alert('Something needs attension');
$r->log->crit('A critical error');
$r->log->error('Something went wrong');
$r->log->warn('You might want to know...');
$r->log->notice('Take note');
$r->log->info('For your information...');
$r->log->debug('In foobar loop, no widgits');
```

- Beware Apache `LogLevel` configuration
- Example 33: *logging.html*

Useful techniques

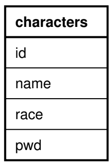
Sending email

- Email is hard
- It's dangerous allow a user-supplied e-mail address on a command line
- Many of the 'special' characters that can cause damage are legal in (some) mail addresses
- Beware 'From:' address vs, envelope return path issues
- Best bet: Use `ppsw.cam.ac.uk` as a smart host, and then use the `Net::SMTP` module
  - See Example 34: *mailer.html*, *send_mail.mason*

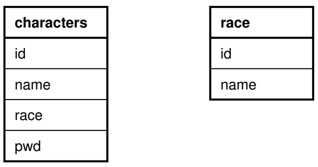
Database interface

- The standard Perl databases interface is DBI
- There are some interesting modules built on this, like Class::DBI, DBIx::Class, DBIx::SearchBuilder, ...
- Load Apache::DBI for persistent database connections

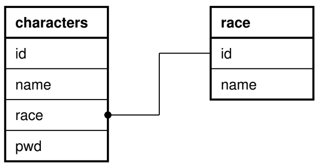
The character table



The race table



Relationship



The program

- See Example 35: *lotr.html*

Raven and lookup

- If a page is Raven-protected, Remote User contains CRSid
  - `$ENV{REMOTE_USER}`
  - `$r->connection->user()` (Apache 1)
  - `$r->user()` (Apache 2)
- CRSid can be looked up in the directory
  - with `Net::LDAP`
  - or with `Ucam::Directory`
- See Example 36: *lookup.html*

Dynamic pages and caching

- Expect caching
  - local browser caching
  - shared caches, configured and transparent
- An issue for authors of dynamic pages when
  - things are not cached when they should be
  - things are cached when they shouldn't
- 9 out of 10 dynamic programs don't express a preference
- This often means that browsers will cache pages (a bit) and shared caches will not, but YMMV
- Different caches and browsers do different things, sometimes for different types of file or types of access
- Avoid making essentially-static contact uncachable
  - for your users
  - for your server
  - for search engines

Controlling caching

- It's all in the headers
- META tags are normally only seen by browsers
- Distinguish between Request and Response headers in standards
- `Pragma: no-cache` probably doesn't work

If you positively don't want a document cached

- Try `Cache-control: no-cache`
- and/or `Expires` in the past

```
Expires: Fri, 30 Oct 1998 14:19:41 GMT
```

If you do want a document cached

- Send `Expires` if possible
- or something like `Cache-control: max-age=86400`
- Consider sending `Last-modified` and/or `ETag`
- ... but what's 'Last modified'?
- Beware of allowing something to be cached if the same URL could produce different output
- Beware of setting `Expires` or `max-age` if not appropriate

Closing remarks

Designing web applications

- Small: one or more top-level components
- Medium: multiple top-level components plus supporting component library
- Large: consider View-Model-Controller (VMC) architecture:
  - View displays/formats data
  - Model manages data access, not web-related
  - Controller holds it all together
- Suggested implementation:
  - View: Mason components
  - Model: one or more Perl libraries (modules)
  - Controller: either a Perl module or one or more top-level components and/or dhandlers

Problems, possible solutions

- HTTP interaction model
- Limitations of HTML form controls
- 75% of all web applications is the same
- Possible solutions
  - Browser-side scripting: Java(ECMA)script, Java
  - Plugins: Flash
  - Ajax?
  - Application frameworks

That's All Folks

If you have been, thanks for listening