Anatomy of a Drupal Page Request

Jennifer Hodgdon
(jhodgdon)

Poplar ProductivityWare
(poplarware.com)

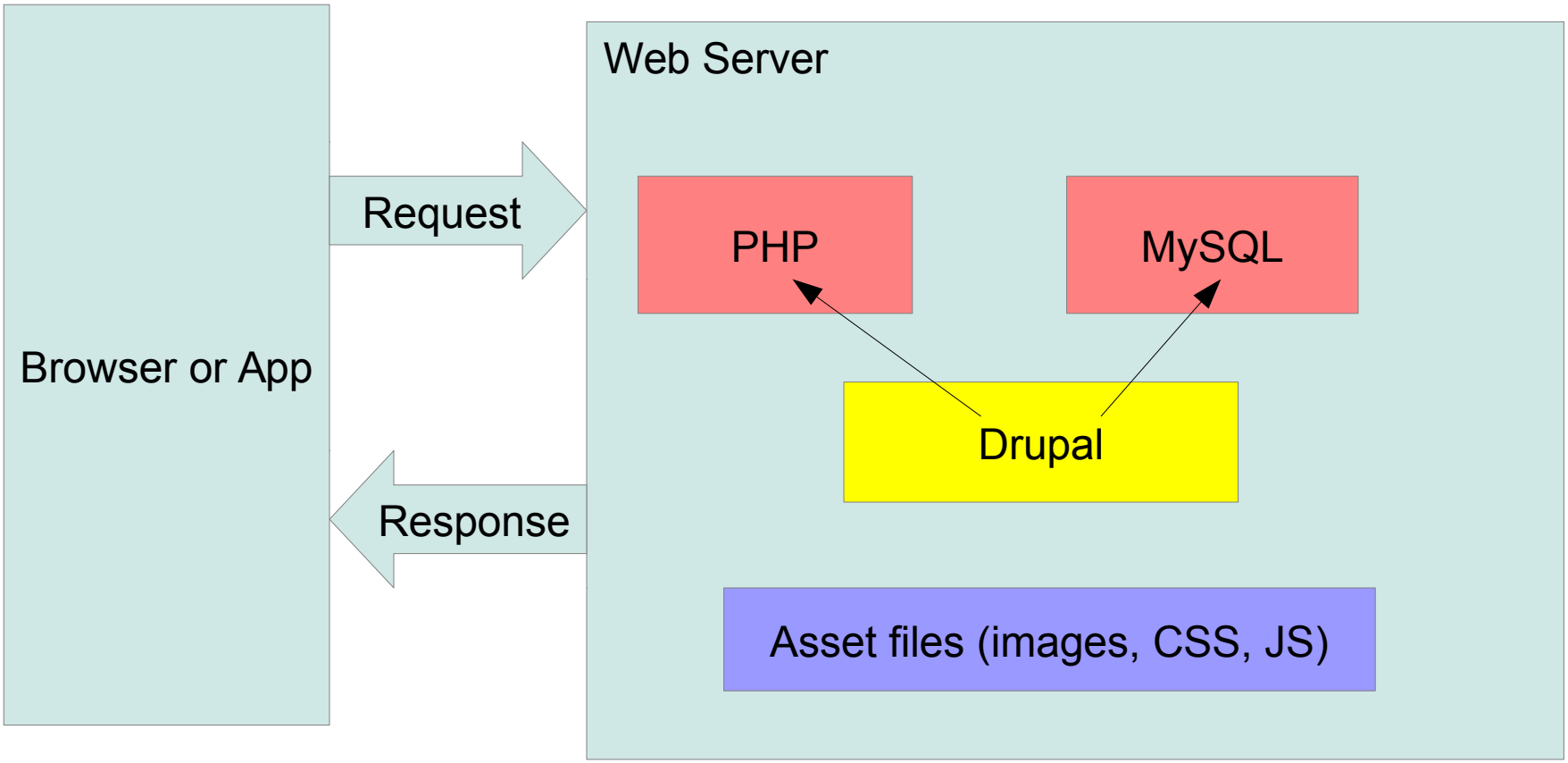
Pacific Northwest Drupal Summit
February 25-26, 2017
Vancouver, BC

Background: Servers and Page/URL Requests

- ***Browser***: Software that you use to request and display web pages
- ***Web page***: one main request (usually HTML), and often many sub-requests (for asset files, Ajax responses, …), incorporated into a single page display in the browser
- ***Asset files***: JavaScript, images, CSS files, ...
- ***App***: Application on your desktop or mobile device that uses data from a web site (usually in a machine-readable format)
- ***Web server***: Hardware and software that responds to incoming page/URL/data/file requests

Poplar ProductivityWare

Graphical Overview



Steps in a URL Request

- You visit a URL in your browser, or use an app that needs some data.
- The browser/app converts the URL to a request: IP address (numeric version of base URL) and request data (rest of the URL, form submission data, cookies, session information, etc.)
- The browser/app sends the request to the web server at that IP address.
- If configured, the server checks its cache for a matching request, and sends back cached data if possible.
- **The server processes the request (figures out what data it needs to send back for that request). This is the part that Drupal is involved in. More on this later...**
- The server sends the data back to the browser/app.
- The browser/app processes and displays the data. This may involve sending other requests (to the same or different servers) and incorporating their results.

The Processing Step on the Web Server

- The web server checks access permissions based on server configuration (including *.htaccess* file or equivalent), IP address of the requester, and request data.
- If the request is for a file that exists (HTML, asset file, ...), and access is allowed, the server sends the file back directly.
- If the request is not for a file that exists, the web server checks its configuration (including *.htaccess* file) for a “rewrite rule”. Drupal's *.htaccess* says:

```
RewriteRule ^ index.php [L]
```

- **The server starts up PHP, loads Drupal's *index.php* file (passing in the request data), and lets Drupal calculate the data/page to send back. More on this later...**

Drupal 7 *index.php* Request Handling

- Determine which *settings.php* file to load (*sites/default/settings.php* or other) and load it.
- Initialize database connection and variable system.
- For anonymous user, check page cache and return cached page if possible.
- Initialize PHP session variables.
- Initialize language system. Load all Core ***include files*** and ***enabled modules' .module files.***
- Check to see if the site is off-line, and return off-line message if so.
- Check the routing system [`hook_menu()`]; call registered *page callback* functions (includes form processing) to calculate the page data, including blocks.
- Determine the delivery method to use for the page (HTML, Ajax, etc.), and process the page appropriately. For HTML: load the *theme,* use the theme to render the data into HTML. For Ajax: transform PHP data into JSON format.
- Print the processed output with HTTP headers, which sends it to the web server.

Poplar ProductivityWare

Drupal 8 Request Graphical Overview

index.php : Initialize autoloader, DrupalKernel, Request

`DrupalKernel::handle($request)` : Load *bootstrap.inc, settings.php*. Initialize Container. Start session. Check middleware for early return (page cache, IP ban, etc.).

`DrupalKernel::preHandle()` : Load include files and *.module* files.

Symfony request handling : Determine controller. Dispatch events.

Module's controller method : Generate response data

Theme system : Render response data

Drupal 8 *index.php* Request Handling

In Drupal 8, many of the steps are handled by outside libraries:

- Start PHP class loader (*autoload.php* from Composer).
- Initialize a *DrupalKernel object.*
- Build a Symfony *Request* object from the server request data (it holds the request data in a usable format).
- **Call the `handle($request)` method on the DrupalKernel object (processes the request). More on this later...**
- Call the `send()` method on the resulting Symfony *Response* object (prints headers and data, sending them to the web server), and terminate.

Drupal 8 DrupalKernel Request Handling

- Load the core *bootstrap.inc* file.
- Determine which *settings.php* file to load (*sites/default/settings.php* or other?) and load it.
- Initialize the *Dependency Injection Container* (an object that keeps track of which *service classes* are used for each *service* within Drupal).
- Initialize PHP session variables.
- Initialize an *HTTP Kernel* object from the "*http_kernel*" service. This is actually a *stacked* set of *middleware* objects that handle early returns (page cache, banning IP addresses, …) and standard responses.
- Call the `handle($request)` method on the each of the middleware objects.
- If no middleware triggers an early return, call the `preHandle()` method on the DrupalKernel object. This loads include files and enabled modules' *.module* files.
- **Process the page using Symfony's HTTP request handling. More on this later...**
- Return the resulting HTTP response object to the caller.

Drupal 8 Symfony Request Handling

- Basic process: Determine which *controller* has registered with the *routing system* to handle the request, check access using the registered method, and invoke the controller.
- Modules can register *routes* (URLs or URL patterns, access restrictions, and controller methods for each route – like `hook_menu()` in Drupal 7). They also provide the controllers (PHP classes/methods that process the request and return response data). Data is returned in a Response object.
- Drupal Core provides a response class that renders data into HTML using the *theme system,* and other response classes for processing Ajax, JSON, etc.
- Symfony also *dispatches* a series of *events* during request processing: *kernel.request, kernel.controller, kernel.response, kernel.terminate, kernel.exception.*
- Drupal Core and modules can *register* to *listen* (respond) to events and modify the default behavior. Core examples:
  - Authenticate using session cookies
  - Handle maintenance mode
  - Decode path aliases

How to Influence Request Handling

Drupal 7:

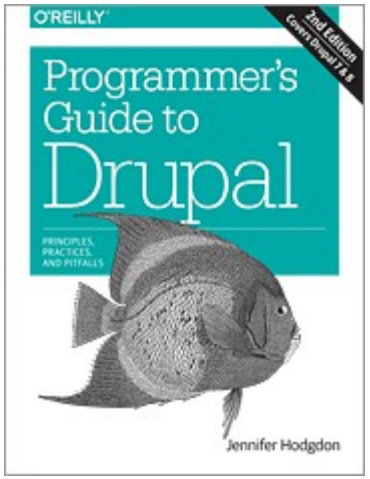
- Implement `hook_menu()` to define a URL route
- Define a page delivery method
- Implement hooks to alter and add to various Core processes
- Theming and theme preprocessing

Drupal 8:

- Define a URL route in a *routing.yml* file or create a dynamic route service
- Define a new Response type
- Register event listeners, create middleware services, implement hooks, define plugins, and override services to alter and add to various Core processes
- Theming and theme preprocessing

Poplar ProductivityWare

Shameless Plug



Programmer's Guide to Drupal, from O'Reilly Media
The second edition covers Drupal 7 and 8.

http://shop.oreilly.com/product/0636920034612.do