Frameworks PHP

Como a "mágica" funciona?

Naylon Kessler de Aquino

www.naylonkessler.com

Co-fundador/CTO do AprovaDETRAN
Co-fundador/CTO da Otimize Growth

Naylon Kessler de Aquino, full stack developer a ~19 anos.

Antes de prosseguirmos

- Não acredite em nada do que eu te disser;
- Não se limite;
- Extrapole os conceitos;
- Não seja “radical”;
- Use a imaginação.

O que veremos

1. Conceito de frameworks;
2. Porquê entender o funcionamento;
3. Partes de um framework PHP;
4. Dependency Injection;
5. Requests;
6. Routing;
7. Responses;
8. Middlewares;
9. Sessões;
10. Views e templates.

O que são frameworks?

Essencialmente frameworks são estruturas de base onde determinadas construções são feitas.

É um conceito presente em diversas áreas e disciplinas.

No desenvolvimento de sistemas, frameworks são arcabouços de software que suportam um desenvolvimento mais ágil e organizado.

Na prática um framework é um conjunto de libraries organizadas pelo uso de uma ou mais arquiteturas, com o propósito de suportar um processo de desenvolvimento.

Classificações ...

- Frameworks de propósito geral:
  - .NET framework
  - Laravel
  - Rails
  - Django
- Frameworks de propósitos específicos?

Porque entender o funcionamento?

- Porque promove o aprendizado mais aprofundado da linguagem usada;
- Porque promove o aprendizado mais aprofundado em arquitetura;
- Porque promove a criação extensões/plugins;
- Porque é um caminho sem volta;
- Porque é divertido :)

Partes de um framework PHP

Console

Scaffolding

Setups

Management

Monitoring

Migrations

Seeding

Data handling

ORM

Query Building

Key/Value storages

Security

Authentication

Authorization

Encryption

Utilities

Storages

Caching

Events

Jobs

Queues

i18n/l10n

Etc ...

Web

Routing

Requests/Responses

Middlewares

Session management

Views

Partes de um framework PHP

Dependency Injection

Era uma uma vez um logger muito simples.

O tempo passou e alguém precisou ...

```php
<?php

namespace Log;

class Logger
{
    protected $whiter;

    public function __construct()
    {
        $this->writer = new TextWriter();
    }

    public function log(string $level, string $message)
    {
        $this->writer->write("[{$level}] {$message}");
    }
}
```

Dependency Injection é uma técnica de implementação de software onde um componente pode prover as dependências a outro componente.

Dependências: quaisquer valores usados por determinado componente.

- Dependency Injection é uma implementação do conceito de Inversion of Control (IoC);
- Pode ser considerado um dos mais importantes conceitos na manutenção do baixo acoplamento entre componentes de software;
- Implementação muito comum nos frameworks.

De repente conseguimos escrever logs em texto simples, XML, JSON e no banco de dados.

```php
<?php

namespace Log;

class Logger
{
    protected $whiter;

    public function __construct(LogWriter $writer)
    {
        $this->writer = $writer;
    }

    public function log(string $level, string $message)
    {
        $this->writer->write("[{$level}] {$message}");
    }
}
```

Na prática, DI é implementada através de um componente de software chamado Dependency Injection Container.

Parece brincadeira mas a base de tudo pode ser resumida nisso:

```php
<?php

class Container
{
    protected $bindings = [];

    public function __get($attribute)
    {
        if (!isset($this->bindings[$attribute])) return;

        $binding = $this->bindings[$attribute];

        return $binding instanceof Closure ? $binding($this) : $binding;
    }

    public function __set($attribute, $value)
    {
        $this->bindings[$attribute] = $value;
    }
}
```

Um exemplo de uso:

```php
<?php

// Create the container
$container = new Container();

// Make bindings
$container->{LogWriter::class} = function ($container) {
    return new JsonWriter();
};
$container->Logger = function ($container) {
    return new Logger($container->{LogWriter::class});
};

// Get an instance
$logger = $container->Logger;
```

Requests

- O atendimento à requisições é parte intrínseca dos frameworks Web;
- No PHP temos acesso aos dados enviados juntos á requisições fazendo uso dos array superglobais;
- $_GET, $_POST, $_FILES, $_REQUEST e $_COOKIE.

As implementações de classes relacionadas a requests nos frameworks PHP geralmente são wrappers em torno dos dados presentes nos array superglobais que adicionam melhorias e utilidades.

```php
class Request
{
    protected $cookies = [];
    protected $payload = [];

    public function __construct(
        array $get = [],
        array $post = [],
        array $files = [],
        array $cookies = []
    ) {
        $this->payload = array_merge($get, $post, $files);
        $this->cookies = $cookies;
    }

    public function cookie(string $name)
    {
        return $this->cookies[$name] ?? null;
    }

    public function input(string $name)
    {
        return $this->payload[$name] ?? null;
    }
}
```

Any suggestion? :-o

Uma classe wrapper de Request ultra simples.

Routing

Alguém se lembra disso:

http://sistemaf*da.com/admin/admin-usuarios.php?acao=listar&pagina=2&ordenar=nome&direcao=asc&filtro=Astrogildo...

- Roteamento é o processo pelo qual uma determinada URL é traduzida em uma ação dentro da aplicação;
- Seu uso se tornou mais ostensivo com o advento do conceito de URLs amigáveis.

As implementações de roteamento nos frameworks PHP são feitas por intermédio da identificação de padrões na URL e na requisição que atendem a determinados critérios pré-estabelecidos. Imagine o processo de resolução de uma RegEx.

/users/**576**/history/**6123578**

/users/**{userId}**/history/**{entryId}**

/^\/users\/**([\d]+?)**\/history\/**([\d]+?)**$/

```php
class Router
{
    protected $routes = [];

    public function method($method, $route, $action)
    {
        $this->routes[$method][$route] = $action;
    }

    public function dispatch($request)
    {
        list($action, $args) = $this->findAction($request->method(), $request->uri());

        return $action($request, ...$args);
    }

    protected function findAction($method, $uri)
    {
        if (empty($this->routes[$method])) return;

        return $this->findFirstAction($method, $uri);
    }

    protected function findFirstAction($method, $uri)
    {
        foreach ($this->routes[$method] as $route => $action) {
            $parsed = $this->parseRoutePatterns($route);
            $matched = $this->matchRoute($uri, $parsed, $action);

            if ($matched) {
                return $matched;
            }
        }
    }

    protected function matchRoute($uri, $route, $action)
    {
        $matched = preg_match('~^'. $route . '$~', $uri, $matches);

        if (!$matched) return;

        return [
            $action,
            array_slice($matches, 1),
        ];
    }

    protected function parseRoutePatterns($route)
    {
        return preg_replace('/(\{.*?\})/', '([^/]+?)', $route);
    }
}
```

Um Router básico

```php
$router = new Router();
$router->method('get', '/', function ($request) {
    return new Response(new View('home'));
});
$router->method('get', '/id/{id}', function ($request, $id) {
    return new Response(new View('id', compact('id')));
});
$response = $router->dispatch(new Request());
$response->send();
```

Exemplo de uso do Router

Responses

- Assim como requests, responses são parte intrínseca dos frameworks Web;
- No PHP (interfaceado com servidor Web) qualquer conteúdo enviado para o stream output é uma response;
- Usa-se os construtos e funções de saída de dados do PHP.

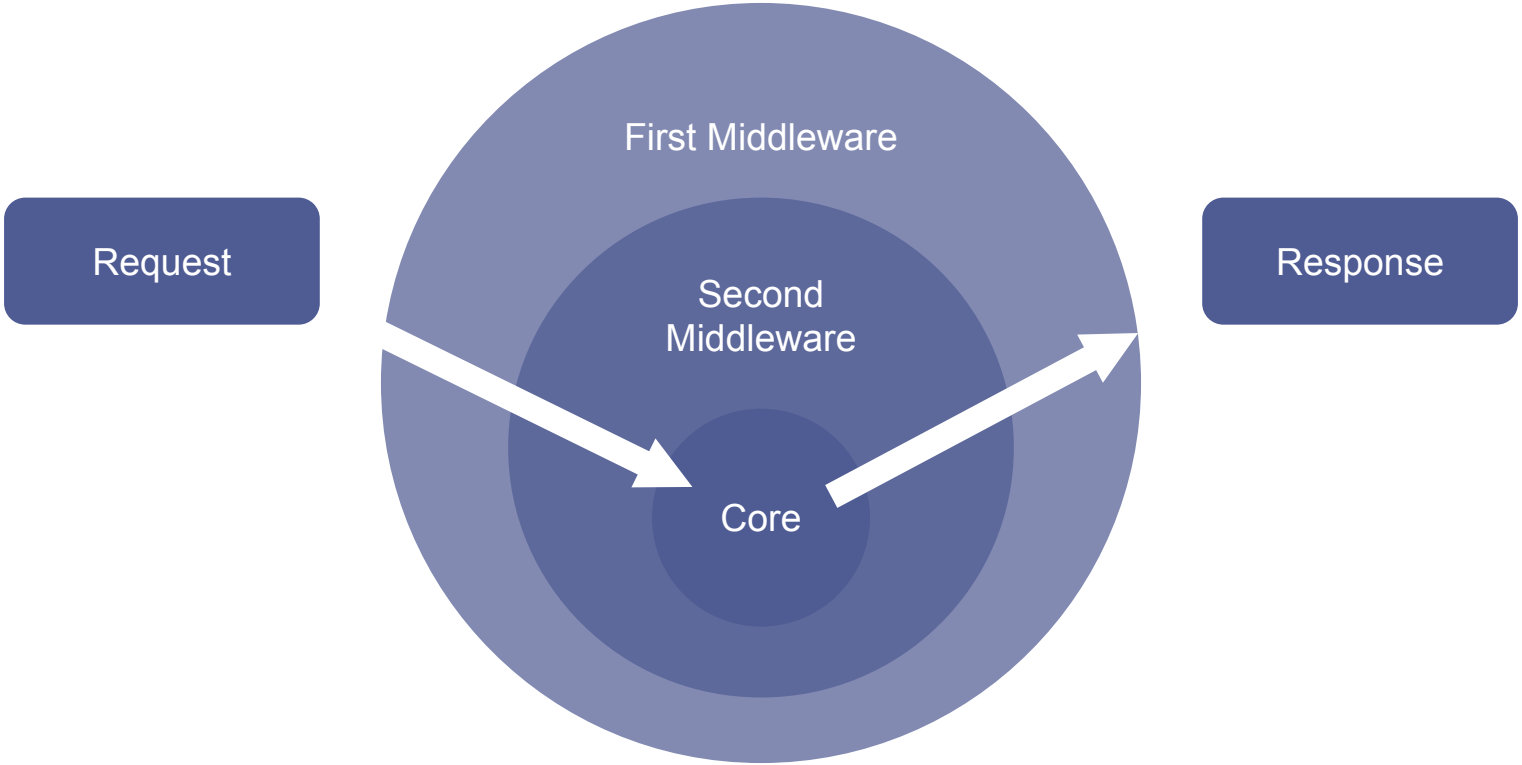
As implementações de responses nos frameworks PHP são geralmente wrappers em torno dos conteúdos enviados para saída. Tais wrappers manipulam, formatam e controlam determinados metadados das responses, a exemplo o Content-Type e o Status Code HTTP.

```php
class Response
{
    protected $content;
    protected $code = 200;
    protected $headers = [];

    public function __construct($content, $code = 200, array $headers = [])
    {
        $this->content = $content;
        $this->code = $code;
        $this->headers = $headers;
    }

    public function send()
    {
        http_response_code($this->code);
        array_walk($this->headers, 'header');

        echo $this->content;
    }
}
```

Uma classe wrapper de Response ultra simples.

WTF are Middlewares

- Middlewares são desafiadores em termos de definição;
- Definição "formal": Middleware é camada de software intermediária que faz interface entre "o núcleo de um SO" e "as aplicações de usuário";
- Definição prática: É uma camada de software que faz a mediação entre duas ou mais partes/processos.



Visão em camadas de um processo usando middlewares.

Nos frameworks Web Middlewares são usados para: transformar os dados e metadados de requests e responses, fazer controle de carga, throttle e bypass, verificações de pré-condições, configurações on-the-fly, dentre outros usos.

```php
class MiddlewareStack
{
    protected $index = 0;

    protected $stack = [];

    public function add($middleware)
    {
        $this->stack[] = $middleware;

        return $this;
    }

    public function run($request)
    {
        $next = $this->getNext();

        if (!$next) return;

        return $next($request, [$this, 'run']);
    }

    protected function getNext()
    {
        return $this->stack[$this->index++] ?? null;
    }
}
```

Um MiddlewareStack básico.

```php
class CheckMaintenanceMiddleware
{
    public function __invoke($request, $next)
    {
        if (app()->isInMaintenance()) {
            return new Response(null, 503);
        }

        return $next($request);
    }
}
```

Um Middleware básico.

Gerenciamento de sessões

- Sessões são usadas para a manutenção de estado entre requisições;
- Os dados são mantidos em arquivos no lado servidor;
- Apenas o identificador da sessão é compartilhado com o lado cliente com um cookie;
- No PHP temos acesso aos dados da sessão pelo acesso ao array superglobal $_SESSION.

As implementações de gerenciamento de sessões nos frameworks PHP podem ser implementações da interface SessionHandlerInterface, wrapper em torno do mecanismo padrão ou podem emular completamente os comportamentos.

```php
class Session
{
    public function __construct(array $options = [])
    {
        if (!headers_sent()) {
            session_start($options);
        }
    }

    public function get(string $key)
    {
        return $_SESSION[$key] ?? null;
    }

    public function set(string $key, $value)
    {
        $_SESSION[$key] = $value;

        return $this;
    }
}
```

Uma classe wrapper de Session ultra simples.

```php
<?php

interface SessionHandlerInterface
{
    public function open($savePath, $sessionName): bool;

    public function close(): bool;

    public function read($id): string;

    public function write($id, $data): bool;

    public function destroy($id): bool;

    public function gc($maxlifetime): bool;
}
```

Overview da interface SessionHandlerInterface

Views, layouts e templates

- Views, layouts e templates exibem as saídas provenientes de processo de roteamento;
- Comumente layouts possuem as partes "estáticas" da renderização enquanto as view, as partes "variáveis";
- Views e layouts podem ainda ser abstrações dos arquivos a serem exibidos;
- Templates são arquivos escritos em uma sintaxe especial e posteriormente compiladas para PHP.

Devido às capacidades de embedding do PHP uma arquivo de view pode ser simplesmente um .php que exibe conteúdos.

```php
<article class="post">
    <header class="post__header">
        <h2><?php echo $post->title; ?></h2>
    </header>
    <section class="post__excerpt">
        <?php echo nl2br($post->excerpt); ?>
    </section>
</article>
```

Um simples arquivo .php com saída de dados.

Linguagens de templating podem ser usadas para promover um código mais limpo, mais rico em features e utilidades, e reduzido em side-effects.

```
<article class="post">
    <header class="post__header">
        <h2>{{ $post->title }}</h2>
    </header>
    <section class="post__excerpt">
        {{ $post->excerpt | nl2br | trim }}
    </section>
</article>
```

Um simples arquivo de template que posteriormente será compilado para PHP.

```php
class TemplateCompiler
{
    protected $patterns = [
        '#\{\{\s?([^\|]*?)\s?\}\}#' => 'compileSimpleOutput',
        '#\{\{\s?(.*?)\s?(\|\s?.+)\s?\}\}#' => 'compileFilteredOutput',
    ];

    public function compile(string $template): string
    {
        return array_reduce(array_keys($this->patterns), function ($result, $pattern) {
            return preg_replace_callback($pattern, [$this, $this->patterns[$pattern]], $result);
        }, $template);
    }

    protected function compileSimpleOutput($match): string
    {
        return '<?php echo ' . $match[1] . '; ?>';
    }

    protected function compileFilteredOutput($match): string
    {
        $filters = array_reverse(array_map('trim', array_filter(explode('|', $match[2]))));
        $openings = array_map([$this, 'filterOpening'], $filters);
        $closing = array_fill(0, count($filters), ')');
        $filters = array_merge($openings, [$match[1]], $closing);

        return '<?php echo ' . implode('', $filters) . '; ?>';
    }

    protected function filterOpening($filter): string
    {
        return $filter . '(';
    }
}
```

Um compilador básico para a "linguagem" de templates concebida.

```php
<article class="post">
    <header class="post__header">
        <h2><?php echo $post->title; ?></h2>
    </header>
    <section class="post__excerpt">
        <?php echo trim(nl2br($post->excerpt)); ?>
    </section>
</article>
```

Resultado do template após compilado.

Referências

MARTIN, Fowler. Inversion of Control Containers and the Dependency Injection pattern. Disponível em <https://martinfowler.com/articles/injection.html>

GAMMA, E.; JOHNSON, R.; HELM, R.; VLISSIDES, J. Padrões de Projetos: Soluções Reutilizáveis. Porto Alegre: Bookman, 2006.

PHP Framework Interop Group - PHP-FIG. Disponível em <https://www.php-fig.org/>

Obrigado

Naylon Kessler de Aquino

www.naylonkessler.com

naylon.kessler@gmail.com