

PSRで学ぶHTTP

Webアプリケーションの実践

2020-12-12 PHPカンファレンス2020 #phpcon



pixiv Inc.
USAMI Kenta

pixiv



USAMI Kenta

@tadsan

お前誰よ

- うさみけんた
- ピクシブ株式会社 pixiv事業本部
 - 現職には2012年から所属
 - PHPは2013年頃から
- Emacs PHP Mode現行メンテナー
 - PHP8対応 Attributeまだです
 - PHPカンファレンス終わったら...
- Emacs JP, php-users-jaなどのコミュニティ



USAMI Kenta

@tadsan

技術など

- 2012年以前はRubyを書いてました
- 現職入社以後は社内APIおよびフレームワークの開発・メンテナンスなどを担当
- いままでのカンファレンスではComposerやPHP静的解析・言語機能などを紹介してきました
- 趣味ではPHPやPHPやEmacs Lispなどを書いてます

今回のお題



うさみけんた



tadsan

PSRはPHP-FIG(PHPフレームワーク相互運用グループ)が発表する勧告群です。
その中ではHTTPについての勧告としてPSR-7, PSR-15, PSR-17そしてPSR-18が発表されており、
これらはWebアプリケーションのモジュール間のインターフェイスとして活用できます。

今回は社内独自のフレームワークをPSR-7/15/17実装として置き換えた事例をとって、PHPと
HTTPリクエストの関係およびPSR-17を実装したWebアプリケーションについて説明します。

このトークを見るにあたって、過去のPHPカンファレンスでの発表を含む「PSR-HTTPシリーズ
を理解するための情報源」を読むことを強く推奨します。

<https://scrapbox.io/php/PSR-HTTPシリーズを理解するための情報源>

PSR-HTTPシリーズを理解するための情報源

リリース済みPSR

- [PSR-7: HTTP Message Interfaces](#) HTTPメッセージクラスの仕様
- [PSR-15: HTTP Handlers](#) HTTPハンドラ(ミドルウェア)の仕様
- [PSR-17: HTTP Factories](#) HTTPメッセージクラスのファクトリの仕様
- [PSR-18: HTTP Client](#) HTTPクライアントクラスの仕様

大前提

PHPとHTTPリクエスト/レスポンスの関係

- [PHP、おまえだったのか。いつもHTTPメッセージを運んでくれたのは。](#)

PSR-7に至る道

- [PHP - 憂鬱な希望としての PSR-7 - Feelin' Kinda Strange](#)

PSR-7はイミュータブルなオブジェクトであること

- [Ps7を使ってみた（というか不変オブジェクトを初めて使った感想）](#)

- 基本的には既存の素晴らしい資料を読めば全部書いてある
 - しかしPHPとHTTPとオブジェクト指向の理解がないと、何が嬉しいのか納得しがたい

- 特にPSR-15に関しては今年の頭にPHPerKaigiのために手を動かしてみるまで、いまいちぴんときていなかった

- 「抽象的な話はわかった(わからん)。で、結局何すればいいの」という理解のギャップを埋めることを目標としています
- 発表後に資料の復習と実践を推奨します

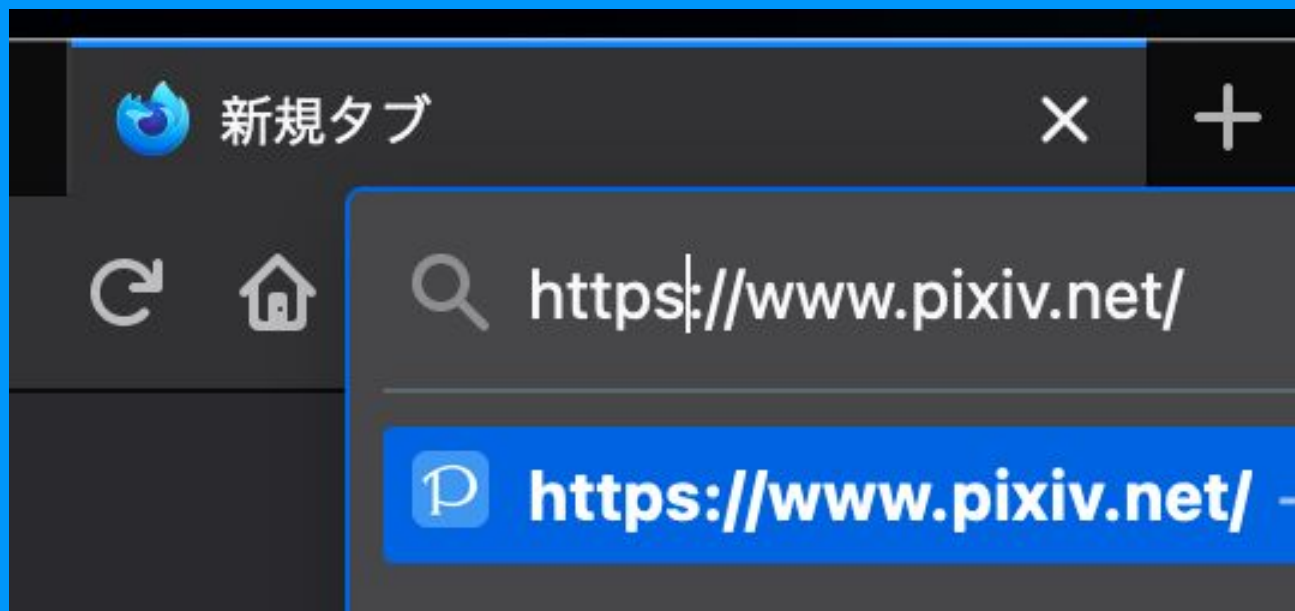
アジェンダ

1. HTTPの概説とPHPの制約
2. HTTPとフレームワークとPSR
3. PSR-7/15/17ベースの実装と勘所

1. HTTPの概説と PHPの制約

ネットの
アドレスのやつ？

HTTPとは何か



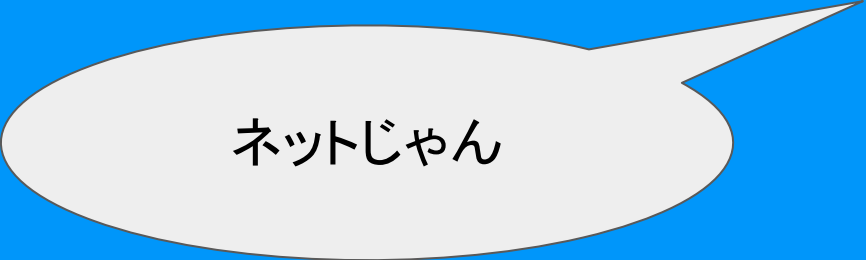
Hypertext Transfer Protocol

出典: フリー百科事典『ウィキペディア (Wikipedia) 』

Hypertext Transfer Protocol（**ハイパーテキスト**・トランスファー・プロトコル、略称 **HTTP**）とは、WebブラウザがWebサーバと通信する際に主として使用する通信プロトコルであり、インターネット・プロトコル・スイートのメンバである。HTMLなどのテキストによって記述されたWebページ等のコンテンツの送受信に用いられる。

https://ja.wikipedia.org/wiki/Hypertext_Transfer_Protocol より引用
最終更新 2020年11月3日 (火) 13:19

- Hypertextって…？



ネットじゃん

- リンクとかクリックすると
別のページに飛べたりする文書
- 要はHTMLとかのこと

Hypertext Transfer Protocol

出典: フリー百科事典『ウィキペディア (Wikipedia) 』

Hypertext Transfer Protocol（**ハイパーテキスト**・トランスファー・プロトコル、略称 **HTTP**）とは、WebブラウザがWebサーバと通信する際に主として使用する通信プロトコルであり、インターネット・プロトコル・スイートのメンバである。HTMLなどのテキストによって記述されたWebページ等のコンテンツの送受信に用いられる。

https://ja.wikipedia.org/wiki/Hypertext_Transfer_Protocol より引用
最終更新 2020年11月3日 (火) 13:19

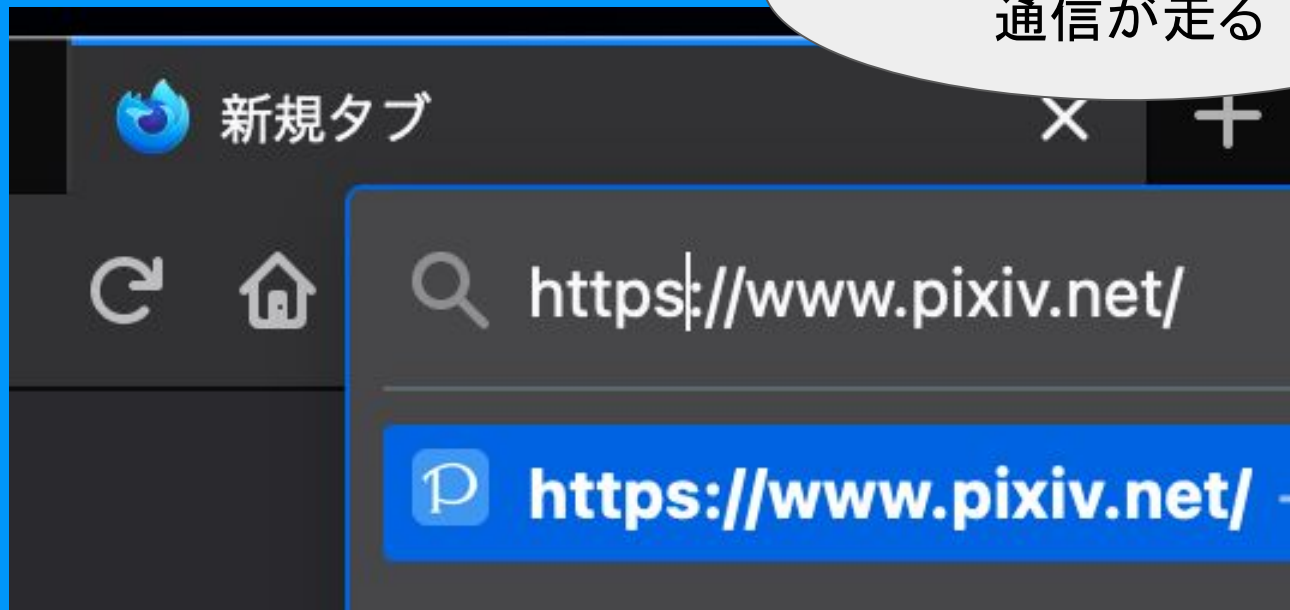
テキストだけじゃない
じゃん

概要 [編集]

HTML (HyperText Markup Language) や XML (Extensible Markup Language) によって記述されたハイパーテキストの転送を主な目的としているが、それ以外にも、バイナリ形式の画像、音声を含め、様々なデータを扱うことが可能である。トランスポート・プロトコルとしてTCPを使用する。

https://ja.wikipedia.org/wiki/Hypertext_Transfer_Protocol より引用
最終更新 2020年11月3日 (火) 13:19

Enterを押すとHTTP
通信が走る



HTTP/3は
違いますが...

- HTTPはTCPベースの通信プロトコル
 - TCPはバイトの並びを送れる
 - HTTP/1はTCPでテキストを
特定のルールで区切って送信する

- 一般的なPHPでのHTTP通信
 - `file_get_contents()` 関数
 - cURL 関数
 - GuzzleやPSR-18で抽象化

- HTTPのやりとりの基本
 - 送信された要求(request)には
返答(response)を返す
 - HTML, CSS, JavaScript, JSON, そのほかなんでも

HTTPリクエスト

HTTPリクエスト

```
POST /post HTTP/1.1
Host: httpbin.org
Accept: application/json
Content-Type: application/json
Connection: close
```

リクエスト行

ヘッダフィールド

```
{"foo": "bar"}
```

リクエストボディ

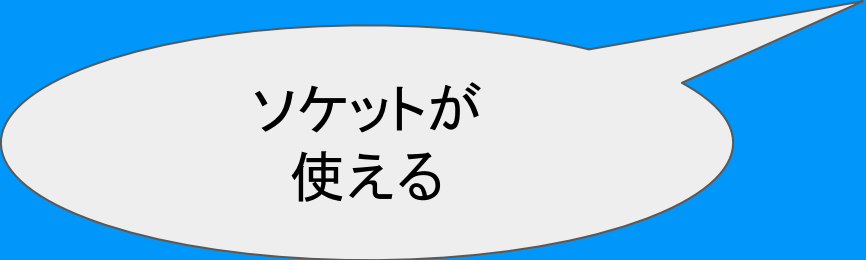
リクエスト行・ヘッダ
の区切りはCR+LF
(¥r¥n)

http://httpbin.org/post



```
POST /post HTTP/1.1
Host: httpbin.org
Accept: application/json
Content-Type: application/json
Connection: close
```

```
{"foo": "bar"}
```



ソケットが
使える

このルール通りにテキストを組み立てて
TCPでいい感じに送ればHTTP通信できる

- `httpbin.org`

- HTTPリクエストに対してさまざまな返答してくれるWebサービス
- ブラウザ上からもテストできる

HTTPリクエスト (リクエスト組み立て)

```
<?php
```

```
$request = implode("\r\n", [  
    'POST /post HTTP/1.1',  
    'Host: httpbin.org',  
    'Accept: application/json',  
    'Content-Type: application/json',  
    'Connection: close',  
    '',  
    'json_encode(['foo' => 'bar'])',  
]);
```

リクエスト行

ヘッダフィールド

リクエストボディ

リクエスト行・ヘッダ
の区切りはCR+LF
(¥r¥n)

HTTPリクエスト (ソケット読み書き)

```
$ip = dns_get_record('httpbin.org', DNS_A);  
$sock = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);  
socket_connect($sock, $ip, 80) or  
die(socket_strerror(socket_last_error()));
```

名前解決

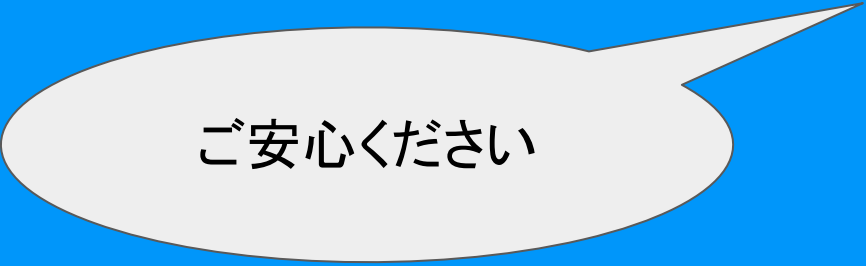
ソケット接続

```
socket_write($sock, $request);
```

ソケット書込(リクエスト)

```
$response = '';  
do {  
    $responses .= ($buf = socket_read($sock, 2048));  
} while ($buf !== '');
```

ソケット読込(レスポンス)



ご安心ください

ところで、この方法で簡単に
通信できるのはHTTP/1.1以下だけ。
HTTPSやHTTP/2では困難。

HTTPSだろうと
ブラウザからは
通信は簡単に見れる

Change language

\$_SERVER

Webブラウザの開発者ツール

Inspector | Console | Debugger | **ネットワーク** | スタイルエディター | パフォーマンス | メモリー | ストレージ | アクセシビリティ | アプリケーション

URL で絞り込み

ステータス	メソッド	ドメイン	ファイル	初期化	タイプ	転送量	サイズ
304	GET	www.php.net	reserved.variables.server.php	browsing-c...	html	キャッシュ	143.20 KB
200	GET	www.php.net	reserved.variables.get.php	other	html	5.89 KB (RCWN)	21.10 KB

すべて | HTML | CSS | JS | XHR | フォント | 画像 | メディア | WS | その他 | キャッシュを無効化 | 帯域制限なし

リファラーポリシー no-referrer-when-downgrade

▼ 応答ヘッダー (531 バイト) 生ヘッダー

- ① cache-control: max-age=0
- ② content-encoding: gzip
- ③ content-language: en
- ④ content-type: text/html; charset=utf-8
- ⑤ date: Fri, 11 Dec 2020 18:13:18 GMT
- ⑥ expires: Fri, 11 Dec 2020 18:13:18 GMT
- ⑦ last-modified: Fri, 11 Dec 2020 12:07:23 GMT
- link: <https://www.php.net/manual/ja/reserved.variables.get.php>; rel=shorturl
- ⑧ server: myracloud
- ⑨ set-cookie: LAST_LANG=ja; expires=Sat, 11-Dec-2021 18:13:17 GMT; Max-Age=31536000; path=/; domain=.php.net
- ⑩ vary: accept-encoding
- X-Firefox-Spdy: h2
- ⑪ x-frame-options: SAMEORIGIN

▼ 要求ヘッダー (422 バイト) 生ヘッダー

- ① Accept: */*
- ② Accept-Encoding: gzip, deflate, br
- ③ Accept-Language: ja,en-US;q=0.7,en;q=0.3
- ④ Connection: keep-alive
- ⑤ Cookie: COUNTRY=NA%2C195.138.250.34; LAST_LANG=ja
- ⑥ Host: www.php.net
- ⑦ Referer: https://www.php.net/manual/ja/reserved.variables.server.php
- ⑧ User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/20100101 Firefox/8.4.0
- X-Moz: prefetch

要求 2 件 | 164.30 KB / 5.89 KB 転送済み | 完了: 3.01 秒 | DOMContentLoaded: 1.67 秒 | load: 2.09 秒

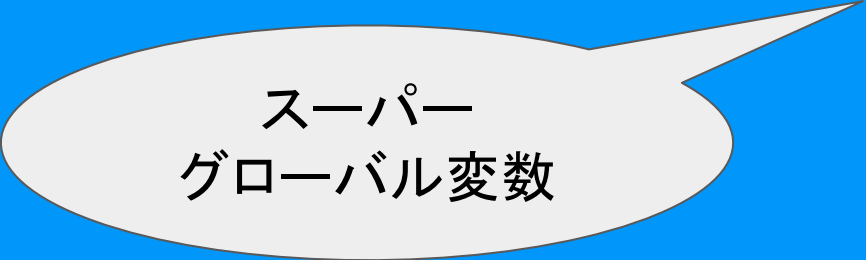
さて

リクエストを送る人が居れば
リクエストを受け取る人も居る

実際にはGoかもしれないしRubyかもしれない…

そうですPHPです

ブラウザでPHPの実行結果
が見えるということはPHPが
HTTPリクエストを受け取って
レスポンスを
返しているにほかならない



スーパー
グローバル変数

\$_SERVER
\$_GET, \$_POST

\$_GET

HTTPリクエスト

```
POST /post?hoge=1&fuga=2 HTTP/1.1
Host: httpbin.org
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Connection: close
```

foo=3&bar=4

\$_SERVER

\$_POST

\$_SERVER

(PHP 4 >= 4.1.0, PHP 5, PHP 7, PHP 8)

\$_SERVER — サーバー情報および実行時の環境情報

説明

`$_SERVER` は、ヘッダ、パス、スクリプトの位置のような情報を有する配列です。この配列のエントリは、Web サーバーにより生成されます。全ての Web サーバーがこれら全てを提供する保障はありません。サーバーは、これらのいくつかを省略したり、この一覧にない他のものを定義する可能性があります。これらの変数の多くは、[» CGI/1.1 specification](#) で定義されています。したがって、これらについては定義されていることを期待することができます。

<https://www.php.net/manual/ja/reserved.variables.server.php> より引用

2020年12月12日 閲覧

- CGI (Common Gateway Interface)
 - かつては掲示板のようなWebアプリ = CGIだった
 - リクエストごとにプロセス起動し標準入出力と環境変数で受け渡し

- CGIから\$_SERVER

- \$_SERVER['REQUEST_URI']
- ヘッダを全て大文字 - を_に変換
- Content-Type ⇒
\$_SERVER['HTTP_CONTENT_TYPE']

- PHPとSAPI (ServerAPI)

- PHPはサーバーの実行環境の差をServerAPIという層で吸収している
- cgi-fcgi, apache, fpm-fcgi, cli-sever...
- 環境ごとにコードを書き換える必要性が限りなく小さい

HTTPレスポンス

HTTPレスポンス

HTTP/1.1 200 OK

バージョン/ステータス

Date: Fri, 11 Dec 2020 16:20:18 GMT

Content-Type: application/json

Content-Length: 339

Connection: close

ヘッダフィールド

Server: gunicorn/19.9.0

Access-Control-Allow-Origin: *

Access-Control-Allow-Credentials: true

{"json": "response..."}

レスポンスボディ

HTTPレスポンス

```
HTTP/1.1 200 OK
Date: Fri, 11 Dec 2020 16:20:18 GMT
Content-Type: text/html
Content-Length: 339
Connection: close
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

<!DOCTYPE html>
<html>
  <head>
```

バージョン/ステータス

ヘッダフィールド

レスポンスボディ

PHPではechoなどで出力したものは
そのままレスポンスボディになる
(レスポンスヘッダは、ボディを
echoする前にheader())を呼んでおく)

PHPでは基本的にこれだけ

```
<?php
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Credentials: true');

echo json_encode($data);
```

PHPでは基本的にこれだけ

```
<?php
header('Content-Type: text/html');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Credentials: true');
?>
<!DOCTYPE html>
<html>
  <head>...
```

PHPでは基本的にこれだけ

```
<?php
header('Content-Type: image/png');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Credentials: true');

// 画像ファイルをそのまま返すこともできる
readfile(__DIR__ . '/../storage/flower.png');
```

- PerlやRubyなどの言語のCGI機能はあくまでもライブラリだが、PHPはServer APIとしてHTTP処理が言語処理系と統合されている

- スーパーグローバル変数
 - `$_GET`, `$_POST`, `$_REQUEST`
`$_SERVER`, `$_COOKIE`, `$FILES`
- グローバル関数
 - `header()`, `setcookie()`, `session`

ならPHPの基本機能をそのまま
使うのが一番自然でいいのでは

そうともいえない事情がある

グローバルは
テストとひたすら相
性が悪い

- スーパーグローバル変数
 - `$_GET`, `$_POST`, `$_REQUEST`
`$_SERVER`, `$_COOKIE`, `$FILES`
- グローバル関数
 - `header()`, `setcookie()`, `session`

- なぜテストと相性が悪いのか
 - 一般的なユニットテストはテスト対象に引数を与えて、その返り値をアサーションする
 - 実装が引数以外のものに依存するとテストしにくい

- とはいえグローバル変数はどうにかなる
 - PHPUnitならテストケース実行前のsetUp()で確実にリセットしておけばいい

- header()/setcookieとCLIの相性
 - PHPUnitのテスト対象で呼ばれるとWarningで止まる
 - Warning: Cannot modify header information

- header()/setcookieとCLIの相性
 - 「特定の場合にヘッダ／Cookieが送信されること」のようなアサーションは書けない
 - ラッパー関数などの工夫で可能
 - 弊社ではStaticMockで対応

2. HTTPとフレームワークとPSR

ここまでの問題点を
整理しよう

- PHPのHTTP機能はグローバルな状態と密結合している
- 操作できないグローバルな状態が存在すると引数と返り値によるアサーションに落とし込めない

それならHTTPを
引数と返り値で扱える
場所に持ってくればいい

HTTPの構造は決まってる
わけじゃないですか

HTTPリクエスト

POST /post HTTP/1.1

リクエスト行

Host: httpbin.org

Accept: application/json

Content-Type: application/json

Connection: close

ヘッダフィールド

{"foo": "bar"}

リクエストボディ

HTTPレスポンス

HTTP/1.1 200 OK

バージョン/ステータス

Date: Fri, 11 Dec 2020 16:20:18 GMT

Content-Type: application/json

Content-Length: 339

Connection: close

ヘッダフィールド

Server: gunicorn/19.9.0

Access-Control-Allow-Origin: *

Access-Control-Allow-Credentials: true

{"json": "response..."}

レスポンスボディ

いかにもクラスに
できそう

システムの奥深くでグローバル
変数を参照されたら困るなら
リクエストオブジェクトを渡して
そこから状態を取得させればいい

header()関数をコールされたら困る
なら、送信したいヘッダを
レスポンスオブジェクトに記録して受
け渡していけばいい

そりゃみんな
やりますよね…

OOP modelling of HTTP

- Cake\Network\{Request, Response}
- CI_Input, CI_Output
- Nette\Http\{Request, Response}
- PHPixie\HTTP\{Request, Responses}
- Symfony\Component\HttpFoundation\{Request, Response}
- yii\web\{Request, Response}
- Zend\Http\{Request, Response}

Rob Allen [HTTP, PSR-7 and Middleware](#)より引用

いろいろあった

2013年にはこういう
のもあった

Stack

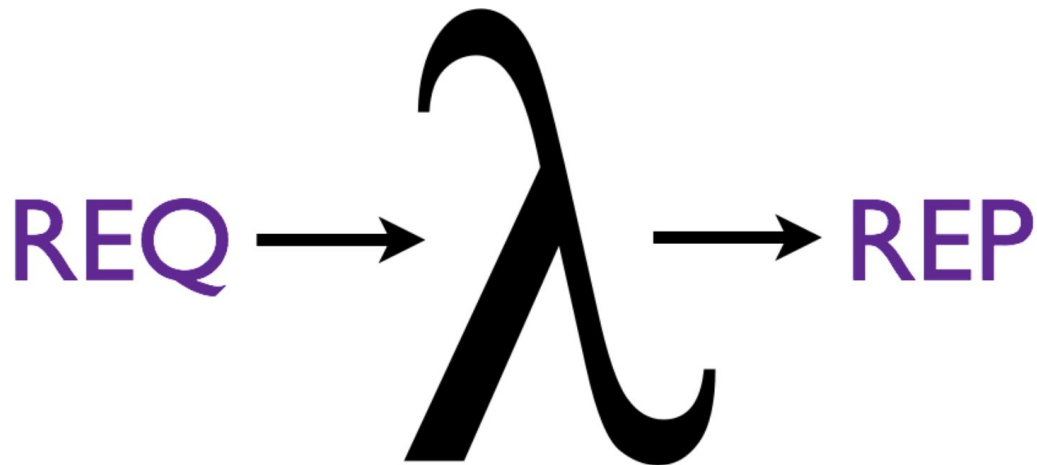
Composing HttpKernelInterface middlewares since 2013!

 stackphp  @stackphp  #stackphp

<https://stackphp.com/> より引用

HttpKernel

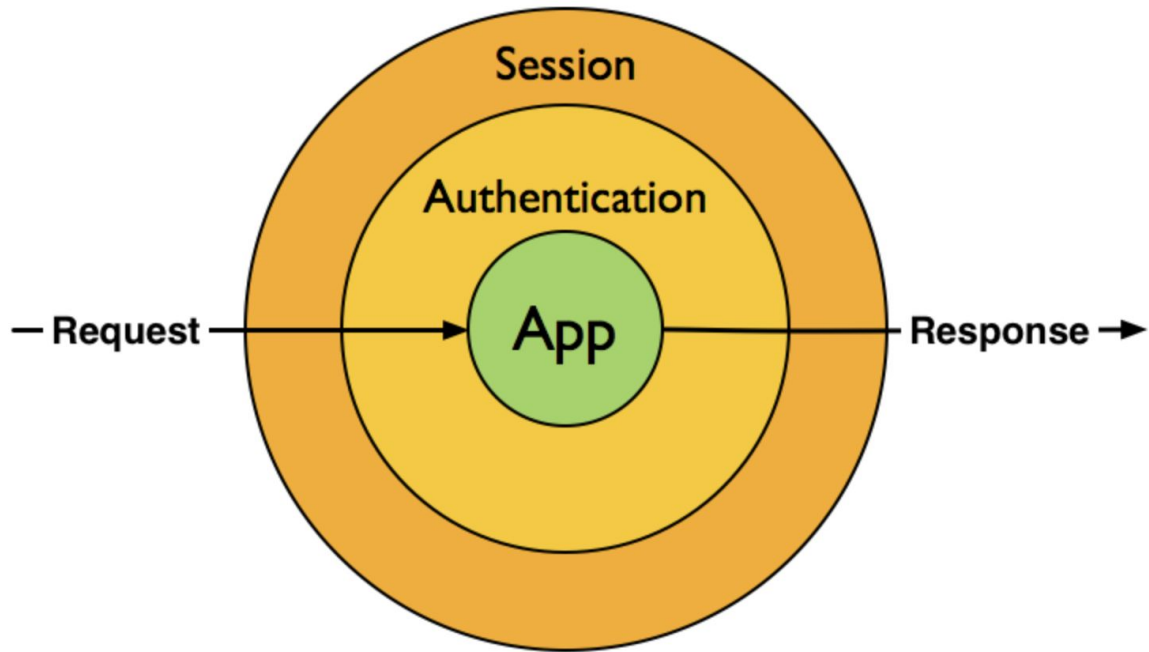
The [HttpKernelInterface](#) models web request and response as PHP objects, giving them value semantics.



<https://stackphp.com/> より引用

What is Stack?

Stack is a convention for composing HttpKernelInterface middlewares.



By wrapping your application in decorators you can add new behaviour from the outside.

そこで出てくるのが
PHP-FIGとPSR

- PHP-FIG
 - PHP Framework Interop Group
(フレームワーク相互運用グループ)
- PSR
 - PHP Standard Recommendation
(PHP標準勧告)

経緯はsasezakiさんの
「憂鬱な希望としての PSR-7」を参照

時は流れて2018年

PSR-15: HTTP Server Request Handlers

PSR-17: HTTP Factories

が受理

これで晴れて具象への依存を避けつつPSR
ベースのWebアプリを書けるようになったとい
う話は、田中ひさてるさんの
PHP-FIGのHTTP処理標準の設計はなぜ
PSR-7/15/17になったのか
を参照ください

3. PSR-7/15/17ベースの実装と勘所

お待たせしました
ここからWebアプリを
作りはじめましょう

- 実装を選ぶ
 - PSRは飽くまでインターフェイス
 - オブジェクトは別に必要

- PSR-7/PSR-17
 - Guzzleもいいがnyholm/psr7を使う
- SAPIとの入出力
 - nyholm/psr7-server
 - narrowspark/http-emitter

- ServerRequestInterface
 - Requestにスーパーグローバル(\$_GETなど)相当+Attributeを付与して引き回せる

- Emitter

- Responseオブジェクトを出力(echo)
- HTTPヘッダに合わせてheader()関数を呼ぶ

プロジェクト初期化

<https://github.com/zonuexe/phpcon-psr-app>

```
% composer require nyholm/psr7 nyholm/psr7-server
% composer require narrowpark/http-emitter

# サーバー起動
% php -S localhost:3939 -t public/ ./public/index.php
```

public/index.php

```
<?php declare(strict_types=1);  
use Narrowspark\HttpEmitter\SapiEmitter;  
use Nyholm\Psr7\Factory\Psr17Factory;  
use Nyholm\Psr7Server\ServerRequestCreator;  
require __DIR__ . '/../vendor/autoload.php';  
  
$psr17Factory = new Psr17Factory();  
$creator = new ServerRequestCreator(  
    $psr17Factory, // ServerRequestFactory  
    $psr17Factory, // UriFactory  
    $psr17Factory, // UploadedFileFactory  
    $psr17Factory // StreamFactory  
);
```

public/index.php

```
$serverRequest = $creator->fromGlobals();

$response = $psr17Factory->createResponse();
    ->withHeader('Content-Type', 'text/plain')
    ->withBody($psr17Factory->createStream('Hello!'));

$emitter = new SapiEmitter();
$emitter->emit($response);
```

- ControllerでResponseを返してみる
 - RequestHandlerで実装するとLaravelのSingle Action Controllerっぽくなる
 - `_invoke()`ではなく`handle()`に

- 注意: 今回はControllerをRequestHandlerにしているが、別の方法で安全に起動できる手段があれば他の方法でもよい

src/Http/RequestHandler/Hello.php

```
final class Hello implements RequestHandlerInterface
{
    private StreamFactory $stream_factory;
    private ResponseFactory $response_factory;

    public function __construct(
        ResponseFactory $response_factory,
        StreamFactory $stream_factory
    ) {
        $this->response_factory = $response_factory;
        $this->stream_factory = $stream_factory;
    }
}
```

src/Http/RequestHandler/Hello.php

```
public function handle(Request $request): Response
{
    return $this->response_factory->createResponse()
        ->withHeader('Content-Type', 'text/plain')
        ->withBody(
            $this->stream_factory->createStream('Hello!')
        );
}
```

public/index.php

```
$serverRequest = $creator->fromGlobals();  
  
$hello = new Hello($psr17Factory, $psr17Factory);  
$response = $hello->handle($serverRequest);  
  
$emitter = new SapiEmitter();  
$emitter->emit($response);
```

- せっかくなので、超簡単な
ルーターも作っていきましょう

public/index.php

```
$router = new StaticRouter([
    '/' => ['GET'=>fn() => new Index($psr17, $psr17)],
    '/hello' => ['GET'=>fn()=> new Hello($psr17, $psr17)],
], fn() => new ErrorPage($psr17Factory, $psr17Factory));

$response = $router->handle($serverRequest);

$emitter = new SapiEmitter();
$emitter->emit($response);
```

src/Http/RequestHandler/StaticRouter.php

```
final class StaticRouter implements
RequestHandlerInterface
{
    private array $routes;
    private Closure $error_page;

    public function __construct(array $routes, Closure
$error_page)
    {
        $this->routes = $routes;
        $this->error_page = $error_page;
    }
}
```

src/Http/RequestHandler/StaticRouter.php

```
public function handle(Request $request): Response
{
    $path = $request->getUri()->getPath();
    $method = $request->getMethod();

    return (
        $this->routes[$path][$method]
        ?? $this->error_page
    )()->handle($request);
}
```

- 既存のミドルウェアを導入していく
 - <https://github.com/middlewares/psr15-middlewares> のプロジェクトがさまざまなミドルウェアを既に実装している

- オブジェクトの生成をどうするか
 - 今回は数が少ないのですべて
 - <https://github.com/phppg/phperkaigi-golf> ではPHP-DIにオブジェクトの生成とディスパッチを任せている

- ServerRequestと型付けの問題
 - `ServerRequest::getParsedBody()`や
`getQueryParams()`から値を取り出す
のは`$_GET`と大差ない
 - `getAttribute`の値も現状
`@var`などで明示的に型付けが必要