PSR

PHP Standards Recommendations

Antonín Neumann

24. 4. 2019

CODING STYLES

- PSR-1: Basic Coding Standard
- PSR-2: Coding Style Guide

AUTOLOADING

- PSR-4: Improved Autoloading

INTERFACES

- PSR-3: Logger Interface
- PSR-6: Caching Interface
- PSR-11: Container Interface
- PSR-13: Hypermedia Links
- PSR-16: Simple Cache
- PSR-14: Event Dispatcher

HTTP

- PSR-7: HTTP Message Interfaces
- PSR-15: HTTP Handlers
- PSR-17: HTTP Factories
- PSR-18: HTTP Client

CODING STYLES

PSR-1: Basic Coding Standard

- pouze UTF-8, pouze LF
- <?php and <?=
- soubor buď deklaruje (třídu, funkce, konstanty) nebo způsobuje side-effects
  - generuje výstup, mění nastavení `.ini`, explicitní použití `require`
- namespace a třídy musí splňovat PSR-0 nebo PSR-4
  - pro PHP 5.3+ musí používat PHP namespaces
- třídy StudlyCase, metody camelCase a konstanty VELKÝMI_PÍSMENY
- visibilita u properties a metod
- názvy properties schválně nejsou zmíněné

PSR-2: Coding Style Guide

- musíte používat/akceptovat PSR-1
- 4 mezery pro odsazení, žádné tabulátory
- true, false, null vše malými písmeny
- žádná mezera na konci řádku
- vždy namespace - minimálně vendor (např. Nette, IW)
- prázdný řádek na konci souboru, žádné `?>` u PHP souborů
- délka řádku - soft limit 120 characters, hard limit nesmí být
- 1 řádek = 1 příkaz
- každý `use` na samostatném řádku

```
namespace IW\Test;

use IW\Whatever;
use IW\WhateverElse as AnotherInterface;

abstract final class Psr2 extends Psr1 implements
     Whatever\SomeInterface,
     AnotherInterface
{
     public $prop;

     public function methodName(int $x, int $y)
     {
          if ($x > $y) {
               //do something
          } elseif ($x < $y) {
               //do something else
          }
     }
}
```

PSR-2: Coding Style Guide

- `extend` a `implements` na stejném řádku jako název třídy
  - seznam více interfaces může být na více řádcích
- property ani metody by neměly být prefixovány podtržítkem
- argumenty u metod nesmí mít mezeru před čárkou a musí ji mít za
- seznam argumentů u metod může být na více řádcích
  - při definici i při volání
- `elseif` psáno dohromady
- u `switch` každý `case` má `break` nebo explicitně `// no break`

squizlabs/PHP_CodeSniffer

FriendsOfPHP/PHP-CS-Fixer

AUTOLOADING

PSR-4: Autoloader

- nahrazuje PSR-0
  - namespace jako název třídy (Vendor_Namespace_Class) již není dovolen
- může být použito s jakýmkoliv dalším autoloadingem
- `\<NamespaceName>(\<SubNamespaceNames>)*\<ClassName>`
  - musí obsahovat top-level namespace (vendor namespace)
  - musí obsahovat název třídy
- na velikosti písmen záleží
- jednotlivé subnamespace reprezentují adresářovou strukturu
- hlavní namespace má definovaný 1 a více výchozích adresářů
- název třídy je stejný jako název souboru + koncovka .php

PSR-4: Autoloader

- autoloader se implementuje pomocí funkce `spl_autoload_register()`
- nejlepší je v dnešní době použít **Composer**
  - ```
    "autoload": {
        "psr-4": {
            "VendorName\\": ["src/", "tests/"]
        },
        "files": ["path/to/extra/file.php"]
    }
    ```
- potom již stačí pouze `require __DIR__ . '/vendor/autoload.php';`
    ```

LOGGING

PSR-3: Logger Interface

- 8 metod podle severity syslogu (RFC 5424)
  - debug, info, notice, warning, error, critical, alert, emergency
- metoda `log($level,$message,array $context)`, musí vrátit stejný výsledek jako metoda pojmenovaná podle severity
  - neznámý level nutí vyhodit exception
- zpráva může obsahovat “proměnné” (placeholders)
  - `{NAME}`, znaky `A-Z`, `a-z`, `0-9`, `_` a `.`
  - escapovat placeholdery by měl implementor
  - hodnoty placeholderů jsou v proměnné $context
  - vyjímka musí být pod klíčem “exception”, pod klíčem “exception” nemusí být nutně vyjímka

PSR-3: Logger Interface

- balíček **psr/log**
- podpůrné třídy a traity
  - `Psr\Log\AbstractLogger` nebo `Psr\Log\LoggerTrait`
    - zjednodušení implementace loggeru, stačí pouze přetížit metodu `log()`
    - plus implementovat nahrazování placeholderů
  - `Psr\Log\LogLevel`
    - enum
  - `Psr\Log\NullLogger`
    - černá díra
  - `Psr\Log\LoggerAwareInterface` nebo `Psr\Log\LoggerAwareTrait`
    - deklaruje způsobilost k používání PSR-3 loggeru

markrogoyski/simplelog-php

monolog/monolog

CACHING

PSR-6: Caching Interface

- kolekce položek v keši *(pool)*
  - hit - položka existuje
  - miss - položka neexistuje, `$item->get()` vrací null
  - deferred - položka nemusí být uložena ihned
- položka *(cache item)*
  - **klíč** - string (`A-Z`, `a-z`, `0-9`, `_`, a `.`), immutable, UTF-8, nejméně 64 znaků
  - **hodnota** - vše co lze serializovat, data musí být vždy vrácena přesně taková jak byla vložena
  - **TTL** *(Time To Live)* - čas mezi uložením položky a jejím zastaráním
  - **expirace** - čas uložení položky + TTL

PSR-16: Simple Cache

- zjednodušení PSR-6 pro běžné použití
- pouze `CacheInterface` (stejné jako *pool*)
  - **klíč** je string (A-Z, a-z, 0-9, _, a .), immutable, UTF-8, nejméně 64 znaků
  - **hodnota** již není obalena do rozhraní, všechny serializovatelné datové typy
    - pokud není možné vrátit přesně stejnou položku jaká byla uložena musí být vrácena *cache miss*
  - **TTL** a **expirace**
    - počet vteřin nebo `DateInterval` objekt
    - pokud není nastaveno default je maximum (ideálně nekonečno)
    - pokud je 0 nebo záporný - položka musí být z cache odstraněna
  - metody `get()`, `set()`, `delete()`, `has()`, `clear()`

DI container

PSR-11: Container Interface

- jen základní abstrakce
  - pouze metody `has()` a `get()`
  - obě vyžadují jako parametr string (identifikátor položky)
- `has()` vrací true pokud je identifikátor pro kontejner známý, jinak false
- metoda `get()` vrací cokoliv (*mixed*) nebo vyhazuje výjimku
  - pokud `has()` vrací false ⇒ `get()` musí vyhodit výjimku
  - NotFoundExceptionInterface
  - dvě po sobě jdoucí úspěšná volání `get()` by měla vrátit stejný objekt (nelze se na to ovšem spolehnout)
- kontejner by se neměl používat jako Service Locator

→

php-di/php-di

pimple/pimple

symfony/dependency-injection

Hypermedia links

PSR-13: Hypermedia Links

- definuje jak by měl vypadat odkaz aby šel snadno sdílet
- balíčky **psr/link** a **fig/link-util**
- *LinkInterface* a *LinkProviderInterface* (pouze pro čtení)
- *EvolvableLinkInterface* a *EvolvableLinkProviderInterface* (immutable)
- odkaz obsahuje target (**href**) a relations (**rel**)
  - relations definuje IANA
  - rel=”nofollow”, rel=”stylesheet”
- ostatní atributy může *serializer* v případě nutnosti vypustit, ale doporučuje se jejich co nejširší podpora

Events

PSR-14: Event Dispatcher

- též "message bus", "hook system"
- umožňuje jednoduše rozšířit funkcionalitu systému
- např: registrace uživatele
  - po uložení dat do databáze odešleme zprávu
  - na tuto zprávu může reagovat několik posluchačů
  - každý z nich něco udělá
    - odešle welcome email uživateli
    - vypočítá procentuální vyplněnost profilu
    - podle preferencí zkusí spárovat nejhodnější pozici ve firmě

PSR-14: Event Dispatcher

- událost (*Event*) - většinou se jedná o nějakou zprávu
- *Stoppable Event* je speciální případ události
  - umožňuje zabránit volání dalších posluchačů
  - má samostatné rozhraní `StoppableEventInterface`
  - metoda `isPropagationStopped()` musí vrátit `true` pokud událost skončila
- vysílač (*Emitter*) - místo v kódu, kde vznikne událost
- posluchač (*Listener*) - volatelný (*callable*) objekt
  - přijímá jako jediný parametr událost
  - vrací `void` a měl by to explicitně vyjádřit pomocí type hint
  - dispatcher musí ignorovat návratovou hodnotu posluchačů
  - posluchač může delegovat práci na jinou část aplikace

PSR-14: Event Dispatcher

- *Dispatcher* - zavolá všechny vyhovující posluchače a předá jim událost
  - dispatcher musí zajistit, že každý posluchač bude zalovaný se stejným objektem události
  - volá posluchače synchronně v pořadí v jakém jsou vrácený providerem
  - vrací stejný objekt události který přijal jako parametr
  - musí počkat na vykonání všech posluchačů
  - pokud pracuje se Stoppable Event musí volat `isPropagationStopped()` před voláním každého posluchače a pokud metoda vrací `true` nesmí být zavolán žádný další posluchač
- *Listener Provider* - rozhoduje kteří posluchači dostanou událost
  - musí dodržet *Liskovové princip zastoupení*
  - může umožňovat registrovat posluchače ve fixním pořadí
  - může generovat seznam posluchačů dopředu
  - může odvozovat posluchače na základě reflexe
  - může volat další providery

→

HTTP

PSR-7: HTTP message interfaces

- standardizace pro request a response
  - `MessageInterface` → `RequestInterface` → `ServerRequestInterface`
- objekty jsou immutable ⟶ metody `with*` a `without*`
- hlavičky - na velikosti písmen nezáleží
  - pouze metoda `getHeaders()` musí vrátit přesně (kvůli maximální zpětné kompatibilitě)
- streamy
  - php://input
- `UploadedFileInterface`
  - normalizace `$_FILES` pole

zendframework/zend-diactoros

slimphp/Slim-Http

guzzle/psr7

PSR-17: HTTP Factories

- slouží pro vytváření PSR-7 objektů
- RequestFactoryInterface
  - `createRequest(string $method, $uri): RequestInterface;`
- ResponseFactoryInterface
  - `createResponse(int $code = 200, string $reasonPhrase = '')`
    `: ResponseInterface;`
- ServerRequestFactoryInterface
  - `createServerRequest(string $method, $uri, array $serverParams = [])`
    `: ServerRequestInterface;`
- StreamFactoryInterface
- UploadedFileFactoryInterface

PSR-18: HTTP Client

```
$content = file_get_contents('http://intraworlds.com');

                          ----------

$ch = curl_init('http://www.intraworlds.com/');

//return the transfer as a string
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

$output = curl_exec($ch)
curl_close($ch);
```

PSR-18: HTTP Client

- hlavní cíl sjednotit chování klientů
- klient může změnit HTTP požadavek i odpověď
  - musí zajistit konzistenci celého objektu
  - komprese obsahu ⇒ přidat `Content-Encoding` a změnit `Content-Length`
- protože jsou PSR-7 objekty immutable nelze se spolehnout že požadavek předaný klientu je přesně stejný jako, ten který je klientem odeslán
- klient musí zajistit vyřešení více krokových HTTP 1xx odpovědí a sám vrací pouze odpovědí se status 200 a vyšší

PSR-15: HTTP Handlers

- handler zpracovává požadavek a vrací odpověď, může skončit výjimkou
  - RequestHandlerInterface
- middleware spolupracuje na zpracování požadavku a vytvoření odpovědi
  - MiddlewareInterface
- odpověď by měla být generovaná přes prototype nebo factory metodou
- aplikace používající middleware by měla mít komponentu která zachytí výjimky a převede je do odpovědi
  - takže je vždy vrácena nějaká odpověď
- 2 samostatné balíčky
  - psr/http-server-middleware, psr/http-server-handler
- součástí není žádný Dispatcher

→

```
$dispatcher = new Dispatcher([
    //Handle errors
    (new Middlewares\ErrorHandler())->catchExceptions(true),

    //Log the request
    new Middlewares\AccessLog($app->get('logger')),

    //Insert the UUID
    new Middlewares\Uuid(),

    //Disable the search engine robots
    new Middlewares\Robots(false),

    //Handle the route
    new Middlewares\RequestHandler(),
]);

$response = $dispatcher->dispatch(ServerRequestFactory::fromGlobals());
```

github.com/middlewares

utils
awesome-psr15-middlewares


INTRAWORLDS
STRONGER RELATIONS
IW
[DEVELOPERS WANTED]
Join us
https://www.intraworlds.cz/

In the next episode you will see...