なんで、ファイル名とクラス名を揃えるの？ 知っておきたいautoloadのはなし

PHPカンファレンス関西2024

2024/2/11

赤塚啓紀

自己紹介

名前	赤塚啓紀
所属	株式会社オフショア
仕事	神戸の会社でPHPで医療機関向けの業務支援システムを作っています。
X	あかつか(@aki_artisan)
趣味	散歩、ハイキング、甘いもの
	昨年関西に引っ越してきてきました システム開発歴はだいたい3年です

`Target class [ ... ] does not exist.`

- このようなエラーを見たことはないでしょうか？
- そして、よくわからないけど、ファイル名を直したらうまく動くようになった！という経験はないでしょうか？
- Laravelで開発していた時の私が遭遇したエラーです。

私がやった解決方法

- エラーで検索して、ファイル名とクラス名を揃えるようにしたらうまく動くようになった。
- でも、なんでファイル名とクラス名を揃えると動くようになったのか、よくわかっていませんでした。
- よくわかっていないと、次に同じエラーが出たときにちゃんと治せるか不安ですよね。
- ちゃんと理解しておけば、怖くありません。

というわけで本題です。

知っておきたい
autoloadのはなし

最初に結論から

- なんでファイル名とクラス名を揃えるの？
  - → autoloadのルール(PSR-4)がそうなっているから

autoloadとは

- 未定義のクラス（インターフェース、トレイトも含む）を呼び出したときに、PHPが自動的にクラスの定義を書いたファイルを読み込んでくれるしくみ

ひとつひとつ説明します。

用語の説明

1. PSR-4ってなに？

autoloadのルール

PHP-FIGという団体が決めていて、デファクトスタンダード（事実上の標準）

依存関係を管理するcomposerというツールがこのルールを満たすように
autoloadを実装してくれています

ということは、使う時はこのルールを守れば良いということ！

1. PSR-4ってなに？

具体的には、クラス名（名前空間とクラス名）とファイルパス（ディレクトリ構成とファイル名）を揃えるということです。

→今はわからなくても大丈夫です！後ほどコードを見ながら説明します。

翻訳した記事があるので、興味のある人は読んでみてください。

2. 名前空間ってなに？

- クラス名の前につけることができる名前のこと
- クラスの集まり同士を分けるために使う

2. 名前空間ってなに？

```php
namespace App\Models;

class Person
{
    // ...
}
```

同じ名前空間の時

```php
namespace App\Models;

$person = new Person();
```

別の名前空間の時

```php
$person = new \App\Models\Person();
```

グループ化できるのでまとまりをわかりやすくできるし、Personクラスを他に作ったとしても名前空間が別だと、同じクラス名でも衝突しないようになる。

autoloadの使い方をコードで理解する

フォルダ構成

以下のようなフォルダ構成とします。

```
.
├── public
│   └── index.php
├── src
│   └── Models
│       └── Person.php
├── vendor
├── composer.json
└── composer.lock
```

1. 同じファイルにクラスを定義している場合（autoloadを使わない場合）

```php
<?php
// クラスを使うファイルと同じファイルにクラスを定義している

class Person
{
    public function greet(string $name) : void
    {
        echo 'Hello ' . $name . '!';
    }
}

$person = new Person();
$person->greet('Taro'); // Hello Taro!
```

1. 同じファイルにクラスを定義している場合（autoloadを使わない場合）

- 実行結果

```
$ php public/index.php
Hello Taro!
```

2. requireでクラス定義ファイルを読み込む場合（autoloadを使わない場合）

- `src/Models/Person.php`

```php
<?php
namespace App\Models;

class Person
{
    public function greet(string $name) : void
    {
        echo 'Hello ' . $name . '!';
    }
}
```

2. requireでクラス定義ファイルを読み込む場合（autoloadを使わない場合）

- `public/index.php`

```php
<?php
require __DIR__ . '/../src/Models/Person.php';
// 使う側のファイルからクラスの定義が書いてあるファイルを読み込む

$person = new App\Models\Person();
$person->greet('Taro'); // Hello Taro!
```

- 使うクラスが増えると、

  requireするファイルが増えてしまいます。

3. autoloadを使う場合（composer）

- `src/Models/Person.php`は同じ

```php
<?php
namespace App\Models;

class Person
{
    public function greet(string $name) : void
    {
        echo 'Hello ' . $name . '!';
    }
}
```

3. autoloadを使う場合（composer）

- `public/index.php`

```php
<?php
require __DIR__ . '/../vendor/autoload.php';

$person = new App\Models\Person();
$person->greet('Taro');
```

3. autoloadを使う場合（composer）

- `composer.json`に設定を追加します。

  （新しい名前空間にオートロードを追加する時のみ）

```
{
    "autoload": {
        "psr-4": {
            "App\\": "src/"
        }
    }
}
```

- この設定は、`App`という名前空間を`src`ディレクトリに紐づけるという意味です。

3. autoloadを使う場合（composer）

- この記述を追加した後は、以下のコマンドを実行する必要があります。

```
$ composer dump-autoload
```

- ※同じディレクトリにある他のクラスが読み込めていたら、`composer.json`の設定や`composer dump-autoload`は不要です。

3. autoloadを使う場合（composer）

- 実行結果

```
$ php public/index.php
Hello Taro!
```

3. autoloadを使う場合（composer）

- ファイルを直接指定していなくても`Person`クラスが読み込めています。
- ここでファイル名を間違って、`src/Models/Preson.php`に書いてしまったとすると、autoload時に`Person.php`を探してしまうので、エラーになってしまいます。

ここまでが、autoloadの動きの部分です。

4. autoloadを実現する仕組み

- 次はこの仕組みをどうやって実現しているのかを見ていきます。

4. autoloadを実現する仕組み

- phpのautoloadを使うには`spl_autoload_register`という関数を使って、**「クラスが未定義だったらこれをしてね」**という処理を登録しておきます。
- これをしておかないとPHPは何をして良いかわからず、結果として「クラスが見つからない」というエラーが出ます。

4. autoloadを実現する仕組み

- composerを使う場合は、`vender/autoload.php`や、`vendor/composer/autoload_real.php`にこの処理が書かれています。
- ので、気になったら見てみてください。
- （私にはちょっと難しかったです。）

autoloadを自作する

autoloadを自作する

- せっかくなので、今回はcomposerに頼らず、`spl_autoload_register`を使って動きを確かめてみましょう！

spl_autoload_register

```
spl_autoload_register(function ($class) {
    // requireなどでクラスの定義が書いてあるファイルを読み込む処理
});
```

- `spl_autoload_register`は関数を引数に取る関数
- クラスが未定義だったときに実行して欲しい処理を登録できる
- 登録する関数の引数(`$class`)には、読み込もうとしているクラスの名前空間付きクラス名が入る（`App\Models\Person`）

app/Models/Person.php

```php
<?php
namespace App\Models;

class Person
{
    public function greet(string $name) : void
    {
        echo 'Hello ' . $name . '!' . PHP_EOL;
    }
}
```

public/index.php

```php
<?php
require __DIR__ . '../lib/autoload.php';

$person = new App\Models\Person();
$person->greet('Taro');
```

lib/autoload.php

```php
<?php
spl_autoload_register(function ($class) {
    $prefix = 'App\\'; // トップレベル名前空間
    $base_dir = __DIR__ . '/../src/'; // Appに紐づけるディレクトリ
    $len = strlen($prefix); // トップレベル名前空間の長さ
    $relative_class = substr($class, $len);
    // トップレベル名前空間を除いたクラス名
    $file = $base_dir
        . str_replace('\\', '/', $relative_class)
        . '.php';
    require_once $file;
});
```

実行結果

```
$ php public/index.php
Hello Taro!
```

例：`$class = 'App\Models\Person'`のとき

```php
<?php
spl_autoload_register(function ($class) {
    $prefix = 'App\\';
    $base_dir = __DIR__ . '/../src/';
    $len = strlen($prefix); // 4
    $relative_class = substr($class, $len); // Models\Person
    $file = $base_dir
        . str_replace('\\', '/', $relative_class)
        . '.php'; // ../src/Models/Person.php
    require_once $file;
});
```

なので、ファイル名とクラス名が不一致だと読み込めない

autoloadを自作する

このようにして、`spl_autoload_register`を使ってautoloadを実装できました。

※実用ではcomposerに頼った方が良いです。

知っておくと嬉しいこと

1. 業務で役立つ

エラーが出ても、ちゃんとどうすれば良いかわかった上で対応できるので、問題解決が早くなります。

2. 勉強しているときにサクッとautoloadをかける

本で勉強してみたいときなど、autoloadを正しく設定できると、すぐに本題に入れるようになります。

まとめ

- **autoloadとは**
  - 未定義のクラスを呼び出したときに、PHPが自動的にクラスが定義されているファイルを読み込んでくれるしくみ
- **なんでファイル名とクラス名を揃えるの？**
  - autoloadのルール（PSR-4）がそうなっているから
- **autoloadを使うにはcomposerを使うのが無難**

今日の話はブログにまとめてあるので、
文字で読みたい方はそちらもどうぞ！

Contact

Twitter: @aki_artisan

GitHub: akinoriakatsuka

Ask the speaker きてね！

ご清聴ありがとうございました