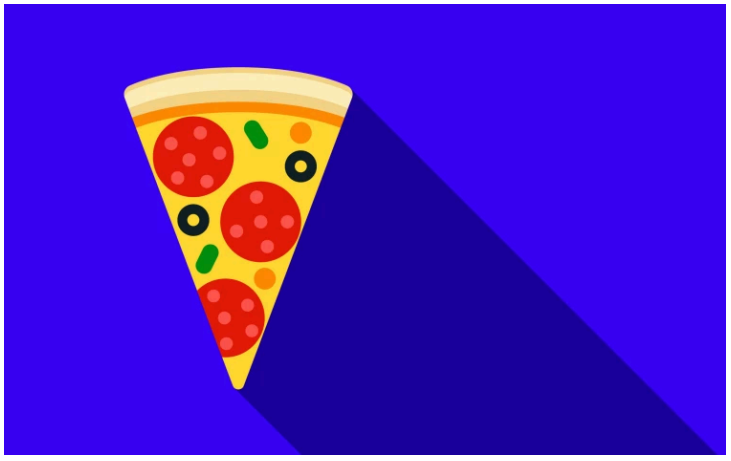
Espacios de nombres PHP en menos de 5 minutos



Chapter 1: Espacios de nombres PHP en menos de 5 minutos

¡Tengo una idea! Dominemos los espacios de nombres de PHP... y hagámoslo en menos de 5 minutos. Toma un café... ¡vamos!

Conoce a Foo

Conoce a `Foo`: una clase de PHP perfectamente aburrida:

Foo.php

```
// ... lines 1 - 2
class Foo
{
    public function doAwesomeThings()
    {

    }
}
```

¡Saluda a `Foo`! Divertidísimo.

Foo.php

```
// ... lines 1 - 2
class Foo
{
    public function doAwesomeThings()
    {
        echo "Hi Foo!\n";
    }
}
```

Para instanciar nuestra nueva clase favorita, me desplazaré a otro archivo y diré - redoble de tambores - `$foo = new Foo()`:

some-other-file.php

```
// ... lines 1 - 2
require 'Foo.php';

$foo = new Foo();
```

¡Tada! Incluso podemos llamar a un método en ella: `$foo->doAwesomeThings()`:

some-other-file.php

```
// ... lines 1 - 2
require 'Foo.php';

$foo = new Foo();

$foo->doAwesomeThings();
```

¿Funcionará? ¡Por supuesto! Puedo abrir un terminal y ejecutar

```
php some-other-file.php
```

Namespaces: Haciendo a Foo más Hipster

¡Ahora mismo, `Foo` no tiene un espacio de nombres! Para que `Foo` sea más hipster, vamos a arreglarlo. Encima de la clase, añade, qué tal, `namespace Acme\Tools`:

Foo.php

```
// ... lines 1 - 2
namespace Acme\Tools;

class Foo
{
// ... lines 7 - 10
}
```

Normalmente el espacio de nombres de una clase coincide con su directorio, pero eso no es técnicamente necesario. ¡Esto lo acabo de inventar yo!

Utilizar una clase con espacio de nombres

¡Enhorabuena! Nuestro amigo `Foo` vive ahora en un espacio de nombres. Poner una clase en un espacio de nombres es muy parecido a poner un archivo en un directorio. Para referenciarla, utiliza la ruta completa y larga de la clase: `Acme\Tools\Foo`:

some-other-file.php

```
// ... lines 1 - 2
require 'Foo.php';

$foo = new \Acme\Tools\Foo();
// ... lines 6 - 8
```

al igual que puedes utilizar la ruta absoluta para referenciar un archivo en tu sistema de archivos:

```
ls /acme/tools/foo
```

Cuando probamos ahora el script

```
php some-other-file.php
```

¡Sigue funcionando!

La declaración de uso mágica y opcional

Y... ¡eso es realmente! Los espacios de nombres son básicamente una forma de... ¡hacer más largos los nombres de tus clases! Añade el espacio de nombres... y luego haz referencia a la clase utilizando el espacio de nombres más el nombre de la clase. Eso es todo.

Pero... ¡tener estos largos nombres de clase justo en medio de tu código es un fastidio! Para solucionarlo, los espacios de nombres de PHP tienen una cosa más especial: la declaración `use`. Al principio del archivo, añade `use Acme\Tools\Foo as SomeFooClass`:

some-other-file.php

```
// ... lines 1 - 2
require 'Foo.php';

use Acme\Tools\Foo as SomeFooClass;
// ... lines 6 - 10
```

Esto crea una especie de... "acceso directo". En cualquier otro lugar de este archivo, ahora podemos escribir simplemente `SomeClassFoo`:

some-other-file.php

```
// ... lines 1 - 2
require 'Foo.php';

use Acme\Tools\Foo as SomeFooClass;

$foo = new SomeFooClass();
// ... lines 8 - 10
```

y PHP sabrá que realmente nos estamos refiriendo al nombre largo de la clase: `Acme\Tools\Foo`.

```
php some-other-file.php
```

O... si omites la parte `as`, PHP asumirá que quieres que este alias sea `Foo`. Así es como suele quedar el código:

some-other-file.php

```
// ... lines 1 - 2
require 'Foo.php';

use Acme\Tools\Foo;

$foo = new Foo();
// ... lines 8 - 10
```

Así pues, los espacios de nombres hacen que los nombres de las clases sean más largos... y las sentencias `use` nos permiten crear atajos para poder utilizar el nombre "corto" en nuestro código.

Clases Básicas de PHP

En el código PHP moderno, casi todas las clases con las que tratas viven en un espacio de nombres... excepto las clases centrales de PHP. Sí, las clases principales de PHP no viven en un espacio de nombres... lo que significa que viven en el espacio de nombres "raíz", como un archivo en la raíz de tu sistema de archivos:

```
ls /some-root-file
```

Juguemos con el objeto del núcleo `DateTime`: `$dt = new DateTime()` y luego `echo $dt->getTimestamp()` con un salto de línea:

some-other-file.php

```
// ... lines 1 - 8
$foo->doAwesomeThings();

$dt = new DateTime();
echo $dt->getTimestamp()."\n";
```

Cuando ejecutamos el script:

```
php some-other-file.php
```

¡Funciona perfectamente! Pero... ahora mueve ese mismo código al método `doAwsomeThings` dentro de nuestro amigo `Foo`:

Foo.php

```php
// ... lines 1 - 2
namespace Acme\Tools;

class Foo
{
    public function doAwesomeThings()
    {
        echo "Hi Foo!\n";

        $dt = new DateTime();
        echo $dt->getTimestamp()."\n";
    }
}
```

Ahora prueba el código:

```
php some-other-file.php
```

¡Ah! ¡Explota! ¡Y comprueba este error!

> *"Clase `Acme\Tools\DateTime` no encontrada"*

El nombre real de la clase debería ser simplemente `DateTime`. Entonces, ¿por qué PHP cree que es `Acme\Tools\DateTime`? Porque los espacios de nombres funcionan como directorios! `Foo` vive en `Acme\Tools`. Cuando decimos simplemente `DateTime`, es lo mismo que buscar un archivo `DateTime` dentro de un directorio `Acme/Tools`:

```
cd /acme/tools
ls DateTime    # /acme/tools/DateTime
```

Hay dos formas de solucionar esto. La primera es utilizar el nombre de clase "totalmente cualificado". Así, `\DateTime`

Foo.php

```
↕ // ... lines 1 - 2
3 namespace Acme\Tools;
4
5 class Foo
6 {
7     public function doAwesomeThings()
8     {
↕ // ... lines 9 - 10
11         $dt = new \DateTime();
↕ // ... line 12
13     }
14 }
```

Sí... eso funciona igual que un sistema de archivos.

```
php some-other-file.php
```

O... puedes usar `DateTime`... y luego eliminar el `\` de abajo

Foo.php

```
↕ // ... lines 1 - 2
3 namespace Acme\Tools;
4
5 use DateTime;
6
7 class Foo
8 {
9     public function doAwesomeThings()
10     {
↕ // ... lines 11 - 12
13         $dt = new DateTime();
↕ // ... line 14
15     }
16 }
```

En realidad es lo mismo: no hay `\` al principio de una declaración `use`, pero debes fingir que lo hay. Esto hace que `DateTime` pase a ser `\DateTime`.

Y... ¡hemos terminado! Los espacios de nombres hacen que los nombres de tus clases sean más largos, las sentencias de uso te permiten crear "atajos" para que puedas utilizar nombres

cortos en tu código y todo el sistema funciona exactamente igual que los archivos dentro de los directorios.

¡Diviértete!

With <3 from SymfonyCasts