

Namespace globale

In PHP, il **namespace** è una funzionalità introdotta in PHP 5.3 che consente di organizzare meglio il codice e di evitare conflitti di nomi tra classi, funzioni e costanti definite in file diversi. Questo è particolarmente utile quando si lavora con grandi progetti o librerie esterne, dove potrebbero esserci classi o funzioni con lo stesso nome.

A cosa serve un namespace?

Immagina di avere due librerie diverse, entrambe contenenti una classe chiamata `Database`. Senza l'uso dei namespace, PHP non saprebbe quale delle due classi `Database` utilizzare e genererebbe un errore. Con i namespace, puoi definire spazi di nomi separati per ciascuna libreria, mantenendo separati i loro contenuti.

Sintassi del namespace

Un namespace si definisce all'inizio di un file PHP con la parola chiave `namespace`. Ad esempio:

```
<?php
namespace MioProgetto\Database;

class Connessione {
    public function connetti() {
        echo "Connessione al database";
    }
}
?>
```

Come utilizzare un namespace

Una volta definito un namespace, è possibile accedere alle classi, funzioni o costanti al suo interno specificando il percorso completo o importandolo con la parola chiave `use` .

Esempio 1: Accesso con il percorso completo

```
<?php
require 'Connessione.php'; // Supponendo che il file Connessione.php contenga il namespace MioProgetto\Database

$databse = new MioProgetto\Database\Connessione();
$databse->connetti();
?>
```

Esempio 2: Uso della parola chiave `use`

```
<?php
require 'Connessione.php';

use MioProgetto\Database\Connessione;

$databse = new Connessione();
$databse->connetti();
?>
```

Esempio completo con più namespace

Immagina di avere due file con classi aventi lo stesso nome, ma provenienti da diverse librerie:

File 1: Library1/Database.php

```
<?php
namespace Library1;

class Database {
    public function get() {
        echo "Library1 Database";
    }
}
?>
```


File 2: Library2/Database.php

```
<?php
namespace Library2;

class Database {
    public function get() {
        echo "Library2 Database";
    }
}
?>
```

File principale: `index.php`

```
<?php
require 'Library1/Database.php';
require 'Library2/Database.php';

use Library1\Database as Db1;
use Library2\Database as Db2;

$db1 = new Db1();
$db1->get(); // Output: Library1 Database

$db2 = new Db2();
$db2->get(); // Output: Library2 Database
?>
```

Conclusione

I namespace in PHP aiutano a risolvere i problemi di conflitti di nomi in progetti più grandi o quando si utilizzano più librerie. Ti permettono di organizzare il codice e facilitano la collaborazione tra moduli o librerie di terze parti.

Autoload delle classi

L'autoloading delle classi in PHP è un meccanismo che consente di caricare automaticamente le classi quando vengono utilizzate, senza la necessità di richiamare manualmente i file con `require` o `include`. Questo diventa particolarmente utile nei progetti di grandi dimensioni, specialmente in combinazione con l'uso dei namespace.

Come funziona l'autoloading?

PHP fornisce diverse funzioni per l'autoloading, tra cui `spl_autoload_register()`, che permette di registrare una o più funzioni di autoload. Quando PHP incontra una classe che non è ancora stata inclusa, invoca automaticamente le funzioni di autoload per cercare di caricarla.

PSR-4: Standard per l'autoloading

In ambito professionale, si usa spesso lo standard **PSR-4** per organizzare i namespace e le classi in modo che corrispondano direttamente alla struttura delle directory. Secondo PSR-4, il nome di una classe (incluso il namespace) determina il percorso del file corrispondente.

Ad esempio, se hai una classe con namespace `Library1\Database`, il file che contiene questa classe dovrebbe trovarsi in una directory che segue questo schema:

```
Library1/Database.php .
```

Autoload con `spl_autoload_register()`

Ecco come implementare l'autoloading usando `spl_autoload_register()` :

1. Struttura delle cartelle:

```
/progetto
├── autoload.php
├── index.php
├── Library1
│   └── Database.php
└── Library2
    └── Database.php
```


2. File `Library1/Database.php` :

```
<?php
namespace Library1;

class Database {
    public function get() {
        echo "Library1 Database";
    }
}
?>
```

3. File `Library2/Database.php` :

```
<?php
namespace Library2;

class Database {
    public function get() {
        echo "Library2 Database";
    }
}
?>
```

4. File `autoload.php` :

Qui creiamo una funzione di autoload che converte il namespace in un percorso di file:

```
<?php
spl_autoload_register(function ($class) {
    // Converte il namespace nel percorso del file
    $file = __DIR__ . '/' . str_replace('\\', '/', $class) . '.php';

    if (file_exists($file)) {
        require_once $file;
    }
});
?>
```

In questa funzione:

- La variabile ``$class`` contiene il nome completo della classe, incluso il namespace.
- La funzione ``str_replace('\\\\', '/', $class)`` converte i backslash del namespace in slash per formare il percorso del file.
- La variabile ``__DIR__`` rappresenta la directory corrente, quindi il percorso completo del file diventa qualcosa come ``/progetto/Library1/Database.php``.

5. File `index.php` :

Ora, possiamo usare le classi senza dover fare manualmente i `require` :

```
<?php
require 'autoload.php'; // Include il nostro file di autoload

use Library1\Database as Db1;
use Library2\Database as Db2;

$db1 = new Db1();
$db1->get(); // Output: Library1 Database

$db2 = new Db2();
$db2->get(); // Output: Library2 Database
?>
```

Cosa succede qui?

1. Quando tentiamo di creare un'istanza di `Db1` o `Db2`, PHP non trova subito le classi, quindi invoca l'autoloader registrato con `spl_autoload_register()`.

2. La funzione di autoload cerca il file corrispondente al namespace e lo include automaticamente.

3. Non c'è più bisogno di fare `require` per ogni file in modo manuale.

Autoloading con Composer

Un'alternativa comune e più automatizzata per l'autoloading è usare **Composer**, un gestore di dipendenze per PHP. Composer può generare automaticamente il codice di autoload per te, seguendo lo standard PSR-4. Per fare ciò, devi semplicemente definire un mapping nel file `composer.json`.

Ecco come sarebbe il file `composer.json` per configurare l'autoload PSR-4:

```
{
    "autoload": {
        "psr-4": {
            "Library1\\": "Library1/",
            "Library2\\": "Library2/"
        }
    }
}
```

Dopo aver definito questo, esegui il comando `composer dump-autoload`, e Composer genererà automaticamente un autoloader che rispetta PSR-4.

Conclusione

Grazie all'autoloading, non devi più includere manualmente ogni file di classe. Questo migliora la gestione e la manutenzione del codice, rendendo più facile lavorare con grandi progetti.

- [PHP namespace](#)