

Có gì mới trong PHP 5.3, Phần 3: Không gian tên

John Mertic

Kỹ sư phần mềm
SugarCRM

06 11 2009

PHP V5.3 được lên lịch để phát hành sớm. Loạt bài "[Có gì mới trong PHP V5.3](#)" này bao gồm các tính năng mới và thú vị trong bản phát hành này. [Phần 1](#) của loạt bài xem xét các thay đổi cho lập trình hướng đối tượng và xử lý đối tượng trong PHP 5.3. [Phần 2](#) xem xét các hàm lambda và các bao đóng. Trong Phần 3 dưới đây, chúng ta xem xét các không gian tên (namespaces), là một trong các tính năng được đoán trước nhiều nhất và gây tranh cãi nhất trong bản phát hành này của PHP. Khái niệm về không gian tên đảm bảo cách trợ giúp tránh các vấn đề về nhiều hàm, lớp, và các hằng của cùng tên được định nghĩa nhiều lần.

[Xem thêm bài trong loạt bài này](#)

Không gian tên tồn tại trong nhiều ngôn ngữ, gồm ngôn ngữ lập trình C++ và Java™. Chúng xuất hiện để giúp đỡ với việc tổ chức các cơ sở mã lớn, nơi thường có một mối lo ngại về chồng chéo hàm hoặc các tên lớp với ứng dụng và các vấn đề mà có việc đó xảy ra. Việc sử dụng các không gian tên có thể giúp nhận ra hàm hoặc công cụ nào mà mã đó cung cấp, hoặc thậm chí giúp xác định nguồn gốc của nó. Một ví dụ về điều này là không gian tên Hệ thống trong C#, có chứa tất cả các hàm và lớp được .NET framework cung cấp.

Trong các ngôn ngữ khác không có các không gian tên chính thức (chẳng hạn như các bản PHP V5.2 và trước đó), những người thường sẽ bắt chước các không gian tên bằng cách cố gắng sử dụng một quy ước đặt tên riêng trong các tên lớp hoặc tên hàm. Zend Framework thực hiện việc này, mỗi tên lớp bắt đầu bằng zend, và mỗi không gian tên con (child namespace) được tách riêng bằng một nét gạch dưới (underscore). Ví dụ, định nghĩa lớp `Zend_Db_Table` là một lớp, bộ phận của Zend Framework và có chức năng cơ sở dữ liệu. Một vấn đề với cách tiếp cận này là ở chỗ mã kết quả có thể trở nên dài dòng (verbose), nhất là khi lớp hoặc hàm đó sâu vài lớp (`Zend_Cache_Backend_Apc` là một ví dụ về việc này từ Zend Framework). Một vấn đề nữa là toàn bộ mã phải phù hợp với kiểu này, có thể khó khăn nếu bạn đang tích hợp mã của bên thứ ba trong một ứng dụng mà không phù hợp với quy ước đặt tên này.

Lịch sử của không gian tên trong PHP là khá vòng vèo. Lúc đầu chúng định dùng làm một bộ phận của PHP V5, nhưng chúng đã bị loại bỏ trong các giai đoạn phát triển do việc thực hiện đúng cách là không thành công. Cuối cùng chúng sống lại như một bộ phận của PHP V6, sau đó chuyển sang PHP V5.3 sau khi quyết định chuyển tất cả các tăng cường không theo unicode vào bản PHP V5.x khác, phát hành vào năm 2007. Trong khi phần lớn trạng thái của các không gian tên là không đổi

do thiết kế độc đáo, vấn đề về việc toán tử nào sẽ sử dụng là vấn đề lớn nhất, với các góc nhìn trái chiều về việc nó sẽ là gì từ tất cả các thành viên của cộng đồng. Một quyết định cuối cùng để sử dụng dấu gạch chéo ngược (\) làm toán tử đã đưa ra vào tháng 10/2008, giải quyết được tất cả các vấn đề mà việc sử dụng các toán tử khác, đã trình bày về thiết kế ngôn ngữ và tính khả dụng.

Các không gian tên trong PHP

Mượn nhiều cú pháp và thiết kế về tên không gian từ các ngôn ngữ khác— nhất là C++. Tuy nhiên, nó xử lý tên không gian khá độc đáo trong một số tình huống, mà có thể là một vật cản đối với những người sử dụng đang mong đợi các tên không gian để làm việc giống hệt như các ngôn ngữ khác. Trong phần này, chúng ta sẽ xem cách các không gian tên làm việc trong PHP.

Định nghĩa một không gian tên

Việc xác định một không gian tên mới là khá bình thường. Để làm như vậy, thêm vào dòng sau đây trong Liệt kê 1 làm lệnh đầu tiên hoặc đầu ra trong một tệp.

Liệt kê 1. Định nghĩa một không gian tên

```
<?php
namespace Foo;
class Example {}
?>
```

Điều quan trọng là phải hiểu rằng việc mô tả trên đây về namespace phải là lệnh đầu tiên hoặc đầu ra trong một tệp. Làm bất kỳ việc nào trước việc này sẽ gây ra một lỗi nghiêm trọng (fatal error). Liệt kê 2 cho thấy một số ví dụ về việc này.

Liệt kê 2. Các cách không tốt khi định nghĩa một không gian tên

```
/* File1.php */
<?php
echo "hello world!";
namespace Bad;
class NotGood {}
?>

/* File2.php */
<?php
namespace Bad;
class NotGood {}
?>
```

Trong phần đầu của Liệt kê 2, chúng ta đã cố gắng phản hồi lại bàn điều khiển trước khi định nghĩa không gian tên, gây nên lỗi nghiêm trọng. Trong phần thứ hai của Liệt kê, chúng ta đã thêm vào một không gian phụ trội trước thẻ mở <?php cũng gây ra một lỗi nghiêm trọng. Điều quan trọng là phải để ý xem đối với tình trạng này khi chúng ta viết mã của chúng ta vì đó là một sơ suất thông thường khi làm việc với các không gian tên trong PHP.

Tuy nhiên, cả hai ví dụ trên đều có thể được viết lại bằng cách tách riêng định nghĩa không gian tên với mã, là một bộ phận mô tả không gian tên vào một tệp riêng của nó, mà có thể được gộp vào trong các tệp ban đầu. Liệt kê 3 trình bày việc này.

Liệt kê 3. Sửa chữa các cách không tốt khi định nghĩa không gian tên

```
/* Good.php */
<?php
namespace Good;
class IsGood() {}
?>

/* File1.php */
<?php
echo "hello world!";
include './good.php';
?>

/* File2.php */
<?php
include './good.php';
?>
```

Bây giờ chúng ta đã nhìn thấy cách định nghĩa một không gian tên đối với mã trong một tệp, chúng ta hãy xem cách vận dụng mã có không gian tên này trong một ứng dụng.

Sử dụng mã không gian tên

Có các tùy chọn khác nhau để gọi ra một hàm, lớp, hoặc hằng số có không gian tên. Có một cách là tham chiếu tường minh không gian tên như là một tiếp đầu ngữ đối với lời gọi. Một tùy chọn khác là định nghĩa một bí danh cho không gian tên và thêm bí danh đó vào đầu lời gọi, nó được thiết kế để làm cho tiền tố không gian tên ngắn hơn. Và cuối cùng, chúng ta có thể chỉ cần sử dụng không gian tên trong phạm vi mã của chúng ta, làm cho nó trở thành không gian tên mặc định và, theo mặc định, làm cho tất cả các lời gọi tham chiếu đến nó. Liệt kê 4 minh họa sự khác nhau giữa các lời gọi.

Liệt kê 4. Gọi một hàm trong một không gian tên

```
/* Foo.php */
<?php
namespace Foo;
function bar()
{
    echo "calling bar....";
}
?>

/* File1.php */
<?php
include './Foo.php';
Foo/bar(); // outputs "calling bar....";
?>

/* File2.php */
<?php
include './Foo.php';
use Foo as ns;
ns/bar(); // outputs "calling bar....";
?>

/* File3.php */
<?php
include './Foo.php';
use Foo;
bar(); // outputs "calling bar....";
?>
```

Liệt kê 4 trình bày các cách khác nhau để gọi ra một hàm `bar()` trong không gian tên `Foo`. Trong tệp `File1.php`, chúng ta trông thấy cách làm tường minh lời gọi, mào đầu cho lời gọi với tên của không gian tên. Tệp `File2.php` sử dụng một bí danh cho tên của không gian tên, nên chúng ta thay tên của không gian tên bằng bí danh đó. Cuối cùng, `File3.php` chỉ cần sử dụng không gian tên, nó cho phép chúng ta thực hiện lời gọi đến `bar()` mà không có bất kỳ tiền tố nào.

Chúng ta cũng có thể định nghĩa nhiều hơn một không gian tên trong một tệp bằng cách thêm vào các lời gọi `namespace` nữa trong tệp. Liệt kê 5 minh họa việc này.

Liệt kê 5. Nhiều không gian tên trong một tệp

```
<?php
namespace Foo;
class Test {}

namespace Bar;
class Test {}

$a = new Foo\Test;
$b = new Bar\Test;

var_dump($a, $b);

Output:
object(Foo\Test)#1 (0) {
}
object(Bar\Test)#2 (0) {
}
```

Bây giờ chúng ta đã có các cơ sở về cách thực hiện một lời gọi bên trong một không gian tên, chúng ta hãy xem xét một số tình huống phức tạp hơn và cách các lời gọi làm việc.

Giải pháp về không gian tên

Một trong những khó khăn đối với các không gian tên là tìm hiểu cách mà giải pháp về phạm vi làm việc. Trong khi các trường hợp đơn giản như trong Liệt kê 4 là có thể hiểu được, vấn đề xuất hiện khi chúng ta bắt đầu lồng các không gian tên vào lẫn nhau, hoặc khi chúng ta ở trong một không gian tên và đang để ý gọi đến vùng toàn cục. PHP 5.3 có các nguyên tắc về việc giải quyết các vấn đề này một cách tự động theo một cách có hiểu biết.

Chúng ta hãy tạo ra một số tệp gộp, tệp nào cũng có hàm `hello()` được định nghĩa trong nó.

Liệt kê 6. Hàm `hello()` được định nghĩa trong các không gian tên khác nhau

```
/* global.php */
<?php
function hello()
{
    echo 'hello from the global scope!';
}
?>

/* Foo.php */
<?php
namespace Foo;
function hello()
{
    echo 'hello from the Foo namespace!';
}
```

```
}
?>

/* Foo_Bar.php */
<?php
namespace Foo/Bar;
function hello()
{
    echo 'hello from the Foo/Bar namespace!';
}
?>
```

Liệt kê 6 xác định hàm `hello()` ba lần trong ba phạm vi khác nhau: trong phạm vi toàn cục, trong không gian tên `Foo`, và trong không gian tên `Foo/Bar`. Tùy thuộc vào phạm vi trong đó lời gọi hàm `hello()` được thực hiện, quyết định được hàm `hello()` nào được gọi ra. Một ví dụ về cách các lời gọi này sẽ trông như trình bày dưới đây. Ở đây, chúng ta sẽ sử dụng không gian tên `Foo` để thấy cách gọi hàm `hello()` như thế nào trong các không gian tên khác.

Liệt kê 7. Gọi ra tất cả các hàm `hello()` từ không gian tên `Foo`

```
<?php
include './global.php';
include './Foo.php';
include './Foo_Bar.php';

use Foo;

hello();           // outputs 'hello from the Foo namespace!'
Bar\hello();       // outputs 'hello from the Foo/Bar namespace!'
\hello();          // outputs 'hello from the global scope!'
?>
```

Chúng ta thấy có thể rút ngắn tiền tố không gian tên khi tham chiếu một không gian tên con trong không gian tên hiện tại (lời gọi `Foo/Bar/hello()` có thể được rút ngắn thành `Bar/hello()`). Và chúng ta trông thấy cách quy định mà chúng ta muốn để gọi ra phương thức trong phạm vi toàn bộ bằng cách chỉ cần mào đầu lời gọi với toán tử không gian tên.

Bây giờ chúng ta đã có cơ chế về cách các không gian tên làm việc, hãy quan sát việc chúng ta có thể sử dụng chúng trong mã của mình.

Các ca sử dụng đối với các không gian tên trong PHP

Mục đích tổng thể đối với các không gian tên là để giúp chúng ta tổ chức tốt hơn mã của chúng ta bằng cách loại bỏ khối lượng các định nghĩa nằm trong phạm vi toàn cục. Trong phần này, chúng ta sẽ xem xét một vài thí dụ có các không gian tên có thể giúp đạt được các mục tiêu này với sự cố gắng nhỏ nhất.

Lập không gian tên của mã bên thứ ba

Nhiều ứng dụng PHP sử dụng mã từ các nguồn khác nhau, dù nó là mã được thiết kế phù hợp với nhau như mã hiện hành trong kho chứa PEAR, mã từ các khung làm việc khác nhau như CakePHP hay Zend Framework, hay là mã khác phát hiện ra từ các địa chỉ khác nhau trên Internet. Một trong những vấn đề lớn nhất khi kết hợp mã này là nó không thể hoà trộn tốt với mã hiện tại; tên các hàm hoặc lớp có thể xung đột với cái mà chúng ta đang sử dụng từ trước trong ứng dụng của chúng ta.

Một thí dụ về việc này là gói PEAR Date. Nó sử dụng tên lớp là Date, đó là một tên lớp khá chung chung và rất có thể đang tồn tại ở đâu đó tại các địa điểm khác trong mã của chúng ta. Như vậy có một cách tốt để khắc phục vấn đề này là thêm một lệnh không gian tên đơn giản vào đầu của tệp Date.php trong gói đó. Bây giờ chúng ta có thể được riêng biệt khi chúng ta muốn sử dụng lớp PEAR Date thay cho lớp của riêng chúng ta.

Liệt kê 8. Sử dụng lớp **PEAR Date** như đã định nghĩa trong một không gian tên

```
<?php
require_once('PEAR/Date.php');

use PEAR\Date;    // the name of the namespace we've specified in PEAR/Date.php

// since the current namespace is PEAR\Date, we don't need to prefix the namespace name
$now = new Date();
echo $now->getDate(); // outputs the ISO formatted date

// this example shows the full namespace specified as part of the class name
$now = new PEAR\Date\Date();
echo $now->getDate(); // outputs the ISO formatted date
?>
```

Chúng ta đã định nghĩa lớp PEAR Date bên trong không gian tên PEAR/Date trong tệp PEAR/Date.php, như vậy mọi điều chúng ta cần làm là gộp mã đó vào tệp của chúng ta và sử dụng không gian tên mới, hoặc đặt tên của không gian tên vào phía trước tên lớp hoặc hàm đó. Với cách này, chúng ta có thể gộp bộ mã của bên thứ ba vào ứng dụng của chúng ta một cách an toàn.

Sự xung đột về tên không chỉ là vấn đề với mã của bên thứ ba. Nó cũng có thể hiện hữu với các cơ sở mã lớn, trong đó có các bộ phận không bao giờ mong đợi đến được gần nhau. Trong phần kế tiếp, chúng ta sẽ xem cách các không gian tên có thể đơn giản hóa tình trạng này như thế nào.

Tránh xung đột về tên hàm tiện ích

Ứng dụng PHP nào cũng có một số phương thức tiện ích. Chúng thực ra không phải là một bộ phận của bất kỳ đối tượng nào trong ứng dụng, và chúng không thực sự bị mất quyền kiểm soát trong bất kỳ nơi nào trong ứng dụng, mặc dù chúng đều đóng một vai trò nào đó cho ứng dụng nói chung. Tuy nhiên chúng có thể gây ra các rắc rối về việc bảo trì khi ứng dụng của chúng ta phát triển lên.

Một nơi mà việc này có thể trở thành vấn đề là kiểm thử mô-đun (unit testing), nơi mà chúng ta viết ra mã để thử nghiệm mã đang chạy một ứng dụng. Phần lớn các bộ kiểm thử mô-đun (unit-testing suites) được thiết kế để chạy từng thử nghiệm một trong toàn bộ bộ kiểm thử. Ví dụ, chúng ta có thể có hai tệp tiện ích mà sẽ không bao giờ gộp vào với nhau, nhưng trong bộ kiểm thử, chúng lại gộp vào nhau, do chúng ta đang kiểm thử toàn bộ ứng dụng cùng một lúc. Trong khi việc thiết kế một ứng dụng bằng cách như vậy không bao giờ là một ý tưởng tốt đối với việc bảo trì dài hạn, thường vẫn tồn tại trong các cơ sở mã kế thừa lớn.

Liệt kê 9 cho biết cách tránh điều này. Chúng ta có hai tệp, `utils_left.php` và `utils_right.php`, là các tập hợp của các hàm tiện ích cho những người sử dụng thuận tay trái và tay phải. Đối với các tệp, chúng ta định nghĩa từng tệp trong không gian tên của chính nó.

Liệt kê 9. utils_left.php và utils_right.php

```
/* utils_left.php */
<?php
namespace Utils\Left;

function whichHand()
{
    echo "I'm using my left hand!";
}
?>

/* utils_right.php */
<?php
namespace Utils\Right;

function whichHand()
{
    echo "I'm using my right hand!";
}
?>
```

Chúng ta định nghĩa một hàm `whichHand()` mà đưa ra kết quả là chúng ta đang sử dụng tay nào. Trong liệt kê 10, chúng ta thấy nó thật dễ dàng khi gộp vào một cách an toàn cả hai tệp và thay đổi giữa hai không gian tên mà chúng ta muốn gọi.

Liệt kê 10. Ví dụ về việc sử dụng cùng nhau các tệp `utils_left.php` và `utils_right.php`

```
<?php
include('./utils_left.php');
include('./utils_right.php');

Utils\Left\whichHand();    // outputs "I'm using my left hand!"
Utils\Right\whichHand();  // outputs "I'm using my right hand!"

use Utils\Left;
whichHand();               // outputs "I'm using my left hand!"

use Utils\Right;
whichHand();               // outputs "I'm using my right hand!"
```

Bây giờ cả hai tệp đều có thể được gộp vào nhau một cách an toàn, và chúng ta quy định không gian tên nào sử dụng để xử lý lời gọi hàm của chúng ta. Và sự ảnh hưởng lên bộ mã hiện thời của chúng ta là nhỏ nhất, do sự tổ chức cải tiến lại để hỗ trợ việc này chỉ yêu cầu chúng ta bổ sung lệnh `use` vào đầu tệp nhằm cho biết dùng không gian tên nào.

Cũng với ý tưởng như vậy có thể được mở rộng ra ngoài bộ mã PHP đã định nghĩa của chúng ta. Trong phần tiếp theo, chúng ta sẽ thấy cách ghi đè lên các hàm trong (internal function) trong một không gian tên.

Ghi đè lên các tên hàm trong

Trong khi các hàm trong trong PHP thường cung cấp tiện ích to lớn, có khi chúng không thực hiện chính xác cái mà chúng ta muốn chúng thực hiện. Chúng ta có thể cần phải gia tăng hành vi của chúng để xứng đáng với cái mà chúng ta muốn hàm đó thực hiện, nhưng chúng ta cũng muốn xác định lại hàm đó bằng một tên nữa nhằm tránh làm lộn xộn phạm vi đó.

Các hàm hệ thống tệp (filesystem functions) là một khu vực mà chúng ta có thể muốn thực hiện việc này. Hãy nói rằng chúng ta muốn chắc chắn rằng bất kỳ tệp nào được tạo ra bằng `file_put_contents()` có tập giấy phép nào đó. Chẳng hạn như hãy nói rằng chúng ta muốn các tệp được tạo ra bằng cái này có thuộc tính chỉ đọc; chúng ta có thể xác định lại hàm đó trong một không gian tên mới như dưới đây.

Liệt kê 11. Định nghĩa `file_put_contents()` bên trong một không gian tên

```
<?php
namespace Foo;

function file_put_contents( $filename, $data, $flags = 0, $context = null )
{
    $return = \file_put_contents( $filename, $data, $flags, $context );

    chmod($filename, 0444);

    return $return;
}
?>
```

Chúng ta gọi ra hàm trong `file_put_contents()` và đặt vào phía trước tên của không gian tên với một dấu chéo ngược để cho biết rằng nó phải được xử lý trong phạm vi toàn cục, nghĩa là hàm trong sẽ được gọi. Sau khi gọi ra hàm trong, thì chúng ta `chmod()` tệp để thiết lập giấy phép thích hợp.

Đây chỉ là một vài thí dụ về các cách chúng ta có thể sử dụng các không gian tên để nâng cao bộ mã của chúng ta. Trong mỗi trường hợp, chúng ta tránh sửa đổi tòi, chẳng hạn như đặt vào phía trước tên của hàm hoặc lớp để làm cho chúng trở thành duy nhất. Chúng ta bây giờ cũng biết việc sử dụng các không gian tên có thể làm cho nó an toàn hơn nhiều như thế nào để gộp vào bộ mã của bên thứ ba trong các ứng dụng lớn, mà không phải lo lắng gì đối với các xung đột về tên.

Tóm tắt

Các không gian tên trong PHP V5.3 là một sự bổ sung vào ngôn ngữ được chào đón, giúp các nhà phát triển tổ chức bộ mã trong một ứng dụng với một cách có hiệu biết. Chúng cho phép bạn tránh được việc sử dụng các tiêu chuẩn đặt tên để xử lý việc lập ra các không gian tên, cho phép bạn viết mã hiệu quả hơn nhiều. Trong khi đã xuất hiện từ lâu, các không gian tên là một sự bổ sung được chào đón đối với bất kỳ ứng dụng PHP quy mô nào gặp phải các xung đột về tên.

Tài nguyên

Học tập

- Bắt đầu loạt bài với "[Cố gì mới trong PHP V5.3, Phần 1: Các thay đổi đến giao diện đối tượng](#)" và "[Phần 2: Closure và các hàm Lambda](#)." Sau đó tiếp tục với "[Cố gì mới trong PHP V5.3, Phần 4: Tạo và sử dụng các lưu trữ Phar](#)."
- Tìm hiểu thêm về [bao đóng](#) tại trang Wikipedia.
- Xem nguyên bản Các yêu cầu bình luận [RFC](#) về các hàm lambda và bao đóng để nhận được chi tiết hơn về cách các bao đóng làm việc như thế nào.
- Đọc "[Một hướng dẫn về di trú PHP V5](#)" để tìm hiểu cách di trú mã được phát triển trong PHP từ V4 đến V5.
- "[Kết nối các ứng dụng PHP với Apache Derby](#)" chỉ cho bạn cách cài đặt và lập cấu hình PHP trên Windows® (một vài bước có thể áp dụng với Linux®).
- [PHP.net](#) là tài nguyên trung tâm dành cho các nhà phát triển PHP.
- Xem trang "[Danh sách các tài liệu PHP nên đọc](#)."
- Duyệt qua toàn bộ [nội dung PHP](#) trên developerWorks.
- Mở rộng các kỹ năng PHP của bạn bằng cách xem trang [Các tài nguyên dự án PHP](#) của IBM developerWorks.
- Lắng nghe các phỏng vấn và thảo luận thú vị cho các nhà phát triển phần mềm, xem trang [developerWorks podcasts](#).
- Sử dụng cơ sở dữ liệu với PHP? Hãy thử [Zend Core for IBM](#), đó là một môi trường phát triển và chạy sản xuất PHP liền khối, sẵn để dùng ngay, dễ dàng cài đặt và có hỗ trợ DB2 V9 của IBM.
- Theo sát [các sự kiện kỹ thuật và các buổi phát tin trên mạng](#).
- Theo dõi các hội nghị sắp được tổ chức, các triển lãm thương mại, các buổi phát thanh trên mạng và [các sự kiện](#) khác trên khắp thế giới được các nhà phát triển mã nguồn mở của IBM quan tâm.
- Truy cập trang [Open source zone](#) của developerWorks để có các thông tin về cách làm thế nào, các công cụ, và cập nhật đối tượng sâu hơn để giúp bạn phát triển các công nghệ mã nguồn mở và sử dụng chúng bằng các sản phẩm của IBM.
- Xem và tìm hiểu về IBM và các công nghệ mã nguồn mở và các chức năng của sản phẩm [các trình diễn miễn phí developerWorks Theo yêu cầu](#).

Lấy sản phẩm và công nghệ

- Đổi mới dự án phát triển mã nguồn mở tiếp theo của bạn bằng [phần mềm chạy thử của IBM](#), có sẵn để tải về hoặc trên DVD.
- Tải về [các phiên bản chạy thử của sản phẩm IBM](#) và bắt tay vào các công cụ phát triển ứng dụng và các sản phẩm phần trung gian từ DB2®, Lotus®, Rational®, Tivoli® và WebSphere®.

Thảo luận

- Tham gia vào [developerWorks blogs](#) và vào cộng đồng developerWorks.
- Tham gia vào [Hội thảo developerWorks PHP Forum: Phát triển các ứng dụng PHP với các sản phẩm Quản trị Thông tin IBM \(DB2, IDS\)](#).

Đôi nét về tác giả

John Mertic

John Mertic đã tốt nghiệp đại học ngành Khoa học Máy tính của Đại học Quốc gia Kent và hiện là kỹ sư phần mềm tại SugarCRM. Ông có nhiều đóng góp vào các dự án mã nguồn mở, đáng kể nhất là các dự án PHP; ông là người tạo ra và duy trì Bộ Cài đặt PHP trên Windows (PHP Windows Installer).

© Copyright IBM Corporation 2009

(www.ibm.com/legal/copytrade.shtml)

[Nhãn hiệu đăng ký](#)

(www.ibm.com/developerworks/vn/ibm/trademarks/)