PHP 魔术常量 PHP PDO

PHP 命名空间(namespace)

PHP 命名空间(namespace)是在PHP 5.3中加入的，如果你学过C#和Java，那命名空间就不算什么新事物。 不过在PHP当中还是有着相当重要的意义。

PHP 命名空间可以解决以下两类问题：

1. 用户编写的代码与PHP内部的类/函数/常量或第三方类/函数/常量之间的名字冲突。
2. 为很长的标识符名称(通常是为了缓解第一类问题而定义的)创建一个别名（或简短）的名称，提高源代码的可读性。

定义命名空间

默认情况下，所有常量、类和函数名都放在全局空间下，就和PHP支持命名空间之前一样。

命名空间通过关键字namespace 来声明。如果一个文件中包含命名空间，它必须在其它所有代码之前声明命名空间。语法格式如下；

```
<?php
// 定义代码在 'MyProject' 命名空间中
namespace MyProject;

// ... 代码 ...
```

你也可以在同一个文件中定义不同的命名空间代码，如：

```
<?php
namespace MyProject;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }

namespace AnotherProject;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
?>
```

不建议使用这种语法在单个文件中定义多个命名空间。建议使用下面的大括号形式的语法。

```
<?php
namespace MyProject {
    const CONNECT_OK = 1;
```

```
    class Connection { /* ... */ }
    function connect() { /* ... */  }
}

namespace AnotherProject {
    const CONNECT_OK = 1;
    class Connection { /* ... */ }
    function connect() { /* ... */  }
}
?>
```

将全局的非命名空间中的代码与命名空间中的代码组合在一起，只能使用大括号形式的语法。全局代码必须用一个不带名称的 namespace 语句加上大括号括起来，例如：

```
<?php
namespace MyProject {

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
}

namespace { // 全局代码
session_start();
$a = MyProject\connect();
echo MyProject\Connection::start();
}
?>
```

在声明命名空间之前唯一合法的代码是用于定义源文件编码方式的 declare 语句。所有非 PHP 代码包括空白符都不能出现在命名空间的声明之前。

```
<?php
declare(encoding='UTF-8'); //定义多个命名空间和不包含在命名空间中的代码
namespace MyProject {

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
}

namespace { // 全局代码
session_start();
$a = MyProject\connect();
echo MyProject\Connection::start();
```

```
}
?>
```

以下代码会出现语法错误：

```
<html>
<?php
namespace MyProject; // 命名空间前出现了“<html>” 会致命错误 -  命名空间必须是程序脚本的第一条语句
?>
```

子命名空间

与目录和文件的关系很像，PHP 命名空间也允许指定层次化的命名空间的名称。因此，命名空间的名字可以使用分层次的方式定义：

```
<?php
namespace MyProject\Sub\Level;  //声明分层次的单个命名空间

const CONNECT_OK = 1;
class Connection { /* ... */ }
function Connect() { /* ... */  }

?>
```

上面的例子创建了常量 MyProject\Sub\Level\CONNECT_OK，类 MyProject\Sub\Level\Connection 和函数 MyProject\Sub\Level\Connect。

命名空间使用

PHP 命名空间中的类名可以通过三种方式引用：

1. **非限定名称，或不包含前缀的类名称，**例如 $a=new foo(); 或 foo::staticmethod();。如果当前命名空间是 currentnamespace，foo 将被解析为 currentnamespace\foo。如果使用 foo 的代码是全局的，不包含在任何命名空间中的代码，则 foo 会被解析为foo。 警告：如果命名空间中的函数或常量未定义，则该非限定的函数名称或常量名称会被解析为全局函数名称或常量名称。

2. **限定名称,或包含前缀的名称，**例如 $a = new subnamespace\foo(); 或 subnamespace\foo::staticmethod();。如果当前的命名空间是 currentnamespace，则 foo 会被解析为 currentnamespace\subnamespace\foo。如果使用 foo 的代码是全局的，不包含在任何命名空间中的代码，foo 会被解析为subnamespace\foo。

3. **完全限定名称，或包含了全局前缀操作符的名称，**例如， $a = new \currentnamespace\foo(); 或 \currentnamespace\foo::staticmethod();。在这种情况下，foo 总是被解析为代码中的文字名(literal name)currentnamespace\foo。

下面是一个使用这三种方式的实例：

file1.php 文件代码

```
<?php
namespace Foo\Bar\subnamespace;

const FOO = 1;
function foo() {}
class foo
{
    static function staticmethod() {}
}
?>
```

file2.php 文件代码

```
<?php
namespace Foo\Bar;
include 'file1.php';

const FOO = 2;
function foo() {}
class foo
{
    static function staticmethod() {}
}

/* 非限定名称 */
foo(); // 解析为函数 Foo\Bar\foo
foo::staticmethod(); // 解析为类 Foo\Bar\foo ，方法为 staticmethod
echo FOO; // 解析为常量 Foo\Bar\FOO

/* 限定名称 */
subnamespace\foo(); // 解析为函数 Foo\Bar\subnamespace\foo
subnamespace\foo::staticmethod(); // 解析为类 Foo\Bar\subnamespace\foo,
                                  // 以及类的方法 staticmethod
echo subnamespace\FOO; // 解析为常量 Foo\Bar\subnamespace\FOO

/* 完全限定名称 */
\Foo\Bar\foo(); // 解析为函数 Foo\Bar\foo
\Foo\Bar\foo::staticmethod(); // 解析为类 Foo\Bar\foo, 以及类的方法 staticmethod
echo \Foo\Bar\FOO; // 解析为常量 Foo\Bar\FOO
?>
```

注意访问任意全局类、函数或常量，都可以使用完全限定名称，例如 \strlen() 或 \Exception 或 \INI_ALL。

在命名空间内部访问全局类、函数和常量：

```
<?php
namespace Foo;

function strlen() {}
const INI_ALL = 3;
class Exception {}

$a = \strlen('hi'); // 调用全局函数strlen
$b = \INI_ALL; // 访问全局常量 INI_ALL
$c = new \Exception('error'); // 实例化全局类 Exception
?>
```

命名空间和动态语言特征

PHP 命名空间的实现受到其语言自身的动态特征的影响。因此，如果要将下面的代码转换到命名空间中，动态访问元素。

example1.php 文件代码：

```
<?php
class classname
{
    function __construct()
    {
        echo __METHOD__,"\n";
    }
}
function funcname()
{
    echo __FUNCTION__,"\n";
}
const constname = "global";

$a = 'classname';
$obj = new $a; // prints classname::__construct
$b = 'funcname';
$b(); // prints funcname
echo constant('constname'), "\n"; // prints global
?>
```

必须使用完全限定名称（包括命名空间前缀的类名称）。注意因为在动态的类名称、函数名称或常量名称中，限定名称和完全限定名称没有区别，因此其前导的反斜杠是不必要的。

动态访问命名空间的元素

```
<?php
namespace namespacename;
class classname
```

```
{
    function __construct()
    {
        echo __METHOD__,"\n";
    }
}
function funcname()
{
    echo __FUNCTION__,"\n";
}
const constname = "namespaced";

include 'example1.php';

$a = 'classname';
$obj = new $a; // 输出 classname::__construct
$b = 'funcname';
$b(); // 输出函数名
echo constant('constname'), "\n"; // 输出 global

/* 如果使用双引号，使用方法为 "\\namespacename\\classname"*/
$a = '\namespacename\classname';
$obj = new $a; // 输出 namespacename\classname::__construct
$a = 'namespacename\classname';
$obj = new $a; // 输出 namespacename\classname::__construct
$b = 'namespacename\funcname';
$b(); // 输出 namespacename\funcname
$b = '\namespacename\funcname';
$b(); // 输出 namespacename\funcname
echo constant('\namespacename\constname'), "\n"; // 输出 namespaced
echo constant('namespacename\constname'), "\n"; // 输出 namespaced
?>
```

namespace关键字和__NAMESPACE__常量

PHP支持两种抽象的访问当前命名空间内部元素的方法，__NAMESPACE__ 魔术常量和namespace关键字。

常量__NAMESPACE__的值是包含当前命名空间名称的字符串。在全局的，不包括在任何命名空间中的代码，它包含一个空的字符串。

__NAMESPACE__ 示例, 在命名空间中的代码

```
<?php
namespace MyProject;

echo '"', __NAMESPACE__, '"'; // 输出 "MyProject"
?>
```

__NAMESPACE__ 示例，全局代码

```
<?php

echo '"', __NAMESPACE__, '"'; // 输出 ""
?>
```

常量 __NAMESPACE__ 在动态创建名称时很有用，例如：

使用__NAMESPACE__动态创建名称

```
<?php
namespace MyProject;

function get($classname)
{
    $a = __NAMESPACE__ . '\\' . $classname;
    return new $a;
}
?>
```

关键字 namespace 可用来显式访问当前命名空间或子命名空间中的元素。它等价于类中的 self 操作符。

namespace操作符，命名空间中的代码

```
<?php
namespace MyProject;

use blah\blah as mine; // see "Using namespaces: importing/aliasing"

blah\mine(); // calls function blah\blah\mine()
namespace\blah\mine(); // calls function MyProject\blah\mine()

namespace\func(); // calls function MyProject\func()
namespace\sub\func(); // calls function MyProject\sub\func()
namespace\cname::method(); // calls static method "method" of class MyProject\cname
$a = new namespace\sub\cname(); // instantiates object of class MyProject\sub\cname
$b = namespace\CONSTANT; // assigns value of constant MyProject\CONSTANT to $b
?>
```

namespace操作符, 全局代码

```
<?php

namespace\func(); // calls function func()
namespace\sub\func(); // calls function sub\func()
namespace\cname::method(); // calls static method "method" of class cname
```

```
$a = new namespace\sub\cname(); // instantiates object of class sub\cname
$b = namespace\CONSTANT; // assigns value of constant CONSTANT to $b
?>
```

使用命名空间：别名/导入

PHP 命名空间支持 有两种使用别名或导入方式：为类名称使用别名，或为命名空间名称使用别名。

在PHP中，别名是通过操作符 use 来实现的. 下面是一个使用所有可能的三种导入方式的例子：

1、使用use操作符导入/使用别名

```
<?php
namespace foo;
use My\Full\Classname as Another;

// 下面的例子与 use My\Full\NSname as NSname 相同
use My\Full\NSname;

// 导入一个全局类
use \ArrayObject;

$obj = new namespace\Another; // 实例化 foo\Another 对象
$obj = new Another; // 实例化 My\Full\Classname  对象
NSname\subns\func(); // 调用函数 My\Full\NSname\subns\func
$a = new ArrayObject(array(1)); // 实例化 ArrayObject 对象
// 如果不使用 "use \ArrayObject" ，则实例化一个 foo\ArrayObject 对象
?>
```

2、 一行中包含多个use语句

```
<?php
use My\Full\Classname as Another, My\Full\NSname;

$obj = new Another; // 实例化 My\Full\Classname 对象
NSname\subns\func(); // 调用函数 My\Full\NSname\subns\func
?>
```

导入操作是在编译执行的，但动态的类名称、函数名称或常量名称则不是。

3、导入和动态名称

```
<?php
use My\Full\Classname as Another, My\Full\NSname;

$obj = new Another; // 实例化一个 My\Full\Classname 对象
$a = 'Another';
```

```
$obj = new $a;      // 实际化一个 Another 对象
?>
```

另外，导入操作只影响非限定名称和限定名称。完全限定名称由于是确定的，故不受导入的影响。

4、导入和完全限定名称

```
<?php
use My\Full\Classname as Another, My\Full\NSname;

$obj = new Another; // 实例化 My\Full\Classname 类
$obj = new \Another; // 实例化 Another 类
$obj = new Another\thing; // 实例化 My\Full\Classname\thing 类
$obj = new \Another\thing; // 实例化 Another\thing 类
?>
```

使用命名空间：后备全局函数/常量

在一个命名空间中，当 PHP 遇到一个非限定的类、函数或常量名称时，它使用不同的优先策略来解析该名称。类名称总是解析到当前命名空间中的名称。因此在访问系统内部或不包含在命名空间中的类名称时，必须使用完全限定名称，例如：

1、在命名空间中访问全局类

```
<?php
namespace A\B\C;
class Exception extends \Exception {}

$a = new Exception('hi'); // $a 是类 A\B\C\Exception 的一个对象
$b = new \Exception('hi'); // $b 是类 Exception 的一个对象

$c = new ArrayObject; // 致命错误, 找不到 A\B\C\ArrayObject 类
?>
```

对于函数和常量来说，如果当前命名空间中不存在该函数或常量，PHP 会退而使用全局空间中的函数或常量。

2、 命名空间中后备的全局函数/常量

```
<?php
namespace A\B\C;

const E_ERROR = 45;
function strlen($str)
{
    return \strlen($str) - 1;
}

echo E_ERROR, "\n"; // 输出 "45"
```

```
echo INI_ALL, "\n"; // 输出 "7" - 使用全局常量 INI_ALL

echo strlen('hi'), "\n"; // 输出 "2"
if (is_array('hi')) { // 输出 "is not array"
    echo "is array\n";
} else {
    echo "is not array\n";
}
?>
```

全局空间

如果没有定义任何命名空间，所有的类与函数的定义都是在全局空间，与 PHP 引入命名空间概念前一样。在名称前加上前缀 \ 表示该名称是全局空间中的名称，即使该名称位于其它的命名空间中时也是如此。

使用全局空间说明

```
<?php
namespace A\B\C;

/* 这个函数是 A\B\C\fopen */
function fopen() {
     /* ... */
     $f = \fopen(...); // 调用全局的fopen函数
     return $f;
}
?>
```

命名空间的顺序

自从有了命名空间之后，最容易出错的该是使用类的时候，这个类的寻找路径是什么样的了。

```
<?php
namespace A;
use B\D, C\E as F;

// 函数调用

foo();      // 首先尝试调用定义在命名空间"A"中的函数foo()
            // 再尝试调用全局函数 "foo"

\foo();     // 调用全局空间函数 "foo"

my\foo();   // 调用定义在命名空间"A\my"中函数 "foo"

F();        // 首先尝试调用定义在命名空间"A"中的函数 "F"
```

```
            // 再尝试调用全局函数 "F"

// 类引用

new B();     // 创建命名空间 "A" 中定义的类 "B" 的一个对象
             // 如果未找到，则尝试自动装载类 "A\B"

new D();     // 使用导入规则，创建命名空间 "B" 中定义的类 "D" 的一个对象
             // 如果未找到，则尝试自动装载类 "B\D"

new F();     // 使用导入规则，创建命名空间 "C" 中定义的类 "E" 的一个对象
             // 如果未找到，则尝试自动装载类 "C\E"

new \B();    // 创建定义在全局空间中的类 "B" 的一个对象
             // 如果未发现，则尝试自动装载类 "B"

new \D();    // 创建定义在全局空间中的类 "D" 的一个对象
             // 如果未发现，则尝试自动装载类 "D"

new \F();    // 创建定义在全局空间中的类 "F" 的一个对象
             // 如果未发现，则尝试自动装载类 "F"

// 调用另一个命名空间中的静态方法或命名空间函数

B\foo();     // 调用命名空间 "A\B" 中函数 "foo"

B::foo();    // 调用命名空间 "A" 中定义的类 "B" 的 "foo" 方法
             // 如果未找到类 "A\B" ，则尝试自动装载类 "A\B"

D::foo();    // 使用导入规则，调用命名空间 "B" 中定义的类 "D" 的 "foo" 方法
             // 如果类 "B\D" 未找到，则尝试自动装载类 "B\D"

\B\foo();    // 调用命名空间 "B" 中的函数 "foo"

\B::foo();   // 调用全局空间中的类 "B" 的 "foo" 方法
             // 如果类 "B" 未找到，则尝试自动装载类 "B"

// 当前命名空间中的静态方法或函数

A\B::foo();    // 调用命名空间 "A\A" 中定义的类 "B" 的 "foo" 方法
               // 如果类 "A\A\B" 未找到，则尝试自动装载类 "A\A\B"

\A\B::foo();   // 调用命名空间 "A" 中定义的类 "B" 的 "foo" 方法
               // 如果类 "A\B" 未找到，则尝试自动装载类 "A\B"
?>
```

名称解析遵循下列规则：

1. 对完全限定名称的函数，类和常量的调用在编译时解析。例如 *new \A\B* 解析为类 *A\B*。

2. 所有的非限定名称和限定名称（非完全限定名称）根据当前的导入规则在编译时进行转换。例如，如果命名空间 *A\B\C* 被导入为 *C*，那么对 *C\D\e()* 的调用就会被转换为 *A\B\C\D\e()*。

3. 在命名空间内部，所有的没有根据导入规则转换的限定名称均会在其前面加上当前的命名空间名称。例如，在命名空间 *A\B* 内部调用 *C\D\e()*，则 *C\D\e()* 会被转换为 *A\B\C\D\e()* 。

4. 非限定类名根据当前的导入规则在编译时转换（用全名代替短的导入名称）。例如，如果命名空间 *A\B\C* 导入为C，则 *new C()* 被转换为 *new A\B\C()* 。

5. 在命名空间内部（例如A\B），对非限定名称的函数调用是在运行时解析的。例如对函数 *foo()* 的调用是这样解析的：

    1. 在当前命名空间中查找名为 *A\B\foo()* 的函数

    2. 尝试查找并调用 *全局(global)* 空间中的函数 *foo()*。

6. 在命名空间（例如*A\B*）内部对非限定名称或限定名称类（非完全限定名称）的调用是在运行时解析的。下面是调用 *new C()* 及 *new D\E()* 的解析过程： *new C()*的解析:

    1. 在当前命名空间中查找*A\B\C*类。

    2. 尝试自动装载类*A\B\C*。

    *new D\E()*的解析:

    1. 在类名称前面加上当前命名空间名称变成：*A\B\D\E*，然后查找该类。

    2. 尝试自动装载类 *A\B\D\E*。

    为了引用全局命名空间中的全局类，必须使用完全限定名称 *new \C()*。

← PHP 魔术常量 PHP PDO →

1 篇笔记

写笔记

可以把非限定名称类比为文件名（例如 comment.php）、.限定名称类比为相对路径名（例如 ./article/comment.php）、完全限定名称类比为绝对路径名（例如 /blog/article/comment.php），这样可能会更容易理解。

再添一例：

```
<?php
//创建空间Blog
namespace Blog;
class Comment { }
//非限定名称，表示当前Blog空间
//这个调用将被解析成 Blog\Comment();
$blog_comment = new Comment();
//限定名称，表示相对于Blog空间
//这个调用将被解析成 Blog\Article\Comment();
$article_comment = new Article\Comment(); //类前面没有反斜杆\
```

```
//完全限定名称，表示绝对于Blog空间
//这个调用将被解析成 Blog\Comment();
$article_comment = new \Blog\Comment(); //类前面有反斜杆\
//完全限定名称，表示绝对于Blog空间
//这个调用将被解析成 Blog\Article\Comment();
$article_comment = new \Blog\Article\Comment(); //类前面有反斜杆\

//创建Blog的子空间Article
namespace Blog\Article;
class Comment { }
?>
```

更多内容可参考：PHP命名空间(Namespace)的使用详解

Alex Gump 7个月前 (08-06)