PHP namespace：命名空间

什么是命名空间？从广义上来说，命名空间是一种封装事物的方法，在很多地方都可以见到这种抽象概念。例如，在操作系统中目录用来将相关文件分组，对于目录中的文件来说，它就扮演了命名空间的角色。

举个简单的例子，文件 foo.txt 可以同时在目录 /home/greg 和 /home/other 中存在，但在同一个目录中不能存在两个 foo.txt 文件。另外，在目录 /home/greg 外访问 foo.txt 文件时，我们必须将目录名以及目录分隔符放在文件名之前，例如 /home/greg/foo.txt。这个原理应用到程序设计领域就是命名空间的概念。

命名空间的定义

PHP 中命名空间（namespace）是在 PHP5.3 中加入的，如果你了解过 C++ 的话，那命名空间就不算什么新事物了。不过命名空间在 PHP 当中还是相当重要的。

PHP 命名空间可以解决以下两类问题：

- 用户编写的代码与 PHP 内部的类/函数/常量或第三方类/函数/常量之间的命名冲突；
- 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，以提高源代码的可读性。

1) 定义命名空间（使用关键字 namespace）

虽然任意合法的 PHP 代码都可以包含在命名空间中，但只有类（包括抽象类和 traits）、接口、函数和常量等类型的代码受命名空间的影响。

命名空间的定义需要通过关键字 namespace 来声明，语法格式如下：

```
namespace 命名空间名;
```

【示例】下面我们来演示一下如何定义了命名空间：

```
<?php
    namespace MyProject;    // 定义名为 MyProject 的命名空间。

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

在声明命名空间之前除了用于定义源文件编码方式的 declare 语句外，所有非 PHP 代码（包括空白符）都不能出现在命名空间声明之前。

另外，与 PHP 其它的语言特征不同，同一个命名空间可以定义在多个文件中，即允许将同一个命名空间的内容分割存放在不同的文件中。

2) 定义子命名空间

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

```
namespace App\Model;
namespace App\Controller\Home;
```

【示例】定义一个子命名空间：

```
<?php
    namespace MyProject\Controller\Home;    // 定义名为 MyProject 的命名空间。

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

3) 在同一个文件中定义多个命名空间

在一个文件中也可以定义多个命名空间，在同一文件中定义多个命名空间有两种语法形式，下面通过示例演示一下：

【示例】定义多个命名空间——简单组合语法。

```
<?php
    namespace MyProject;

    const CONNECT_OK = 1;
    class className {
        /* ... */
    }

    namespace AnotherProject;

    const CONNECT_OK = 1;
    class className {
```

```
13.         /* ... */
14.     }
15. ?>
```

【示例】定义多个命名空间——大括号 {} 语法。

```
01. <?php
02.     namespace MyProject{
03.         const CONNECT_OK = 1;
04.         class className {
05.             /* ... */
06.         }
07.     }
08.
09.     namespace AnotherProject{
10.         const CONNECT_OK = 1;
11.         class className {
12.             /* ... */
13.         }
14.     }
15. ?>
```

在实际的编程实践中，并不提倡在同一个文件中定义多个命名空间。定义多个命名空间主要用于将多个 PHP 脚本合并在同一个文件中。在定义多个命名空间时建议使用大括号形式的语法。

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，同时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，如下所示：

```
01. <?php
02.     namespace MyProject{        // 命名空间中的代码
03.         const CONNECT_OK = 1;
04.         class className {
05.             /* ... */
```

```
06.         }
07.     }
08.
09.     namespace {                 // 命名空间外的全局代码
10.         $obj = new MyProject\className();
11.     }
12. ?>
```

使用命名空间：基础

在讨论如何使用命名空间之前，必须了解 PHP 是如何知道要使用哪一个命名空间中的元素的。我们可以将 PHP 命名空间与文件系统作一个简单的类比。在文件系统中访问一个文件有三种方式：

- 相对文件名形式如 foo.txt。它会被解析为 currentdirectory/foo.txt，其中 currentdirectory 表示当前目录。因此如果当前目录是 /home/foo，则该文件名被解析为 /home/foo/foo.txt；
- 相对路径名形式如 subdirectory/foo.txt。它会被解析为 currentdirectory/subdirectory/foo.txt；
- 绝对路径名形式如 /main/foo.txt。它会被解析为 /main/foo.txt。

PHP 命名空间中的元素使用同样的原理。例如，命名空间下的类名可以通过三种方式引用：

- 非限定名称，或不包含前缀的类名称，例如 `$a=new foo();` 或 `foo::staticmethod();` 。如果当前命名空间是 currentnamespace，那么 foo 将被解析为 currentnamespace\foo。如果使用 foo 的代码是全局的，不包含在任何命名空间中的代码，则 foo 会被解析为 foo。
- 限定名称，或包含前缀的名称，例如 `$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。
- 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 currentnamespace\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。

下面是一个使用这三种方式的实例，我们需要两个 PHP 源文件，分别是 demo.php 和 index.php，示例代码如下：

1) demo.php

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

    const FOO = 1;
    function foo() {}
    class foo
    {
        public function demo() {
            echo '命名空间为：Foo\Bar\Demo';
        }
    }
?>
```

2) index.php

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

    const FOO = 2;
    function foo() {
        echo 'Foo\Bar 命名空间下的 foo 函数<br>';
    }
    class foo
    {
        static function demo(){
            echo '命名空间为：Foo\Bar<br>';
```

```
        }
    }

    /* 非限定名称 */
    foo();                  // 解析为 Foo\Bar\foo resolves to function Foo\Bar\foo
    foo::demo();            // 解析为类 Foo\Bar\foo 的静态方法 staticmethod。
    echo FOO.'<br>';        // 解析为常量 Foo\Bar\FOO

    /* 限定名称 */
    Demo\foo();             // 解析为函数 Foo\Bar\Demo\foo
    Demo\foo::demo();       // 解析为类 Foo\Bar\Demo\foo,
                            // 以及类的方法 demo
    echo Demo\FOO.'<br>';   // 解析为常量 Foo\Bar\Demo\FOO

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

运行结果如下：

```
Foo\Bar 命名空间下的 foo 函数
命名空间为：Foo\Bar
2
Foo\Bar\Demo 命名空间下的 foo 函数
命名空间为：Foo\Bar\Demo
1
Foo\Bar 命名空间下的 foo 函数
命名空间为：Foo\Bar
2
```

> 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 等。

使用命名空间：别名/导入

PHP 允许通过别名引用或导入的方式来使用外部的命名空间，这是命名空间的一个重要特征。这有点类似于在类 unix 文件系统中可以创建对其它的文件或目录的符号连接。

使用 use 关键字可以实现命名空间的导入，从 PHP5.6 开始允许导入函数和常量，并为它们设置别名。语法格式如下：

```
use namespace;
```

在 PHP 中，别名是通过操作符 use 与 as 来实现的，语法格式如下：

```
use 命名空间 as 别名;
```

【示例】使用 use 操作符导入和使用别名。

```
01.  <?php
02.      namespace foo;
03.      use My\Full\Classname as Another;
04.
05.      // 下面的例子与 use My\Full\NSname as NSname 相同
06.      use My\Full\NSname;
07.
08.      // 导入一个全局类
09.      use ArrayObject;
10.
11.      // 导入一个函数
12.      use function My\Full\functionName;
13.
14.      // 导入一个函数并定义别名
```

```
15.     use function My\Full\functionName as func;
16.
17.     // 导入一个常量
18.     use const My\Full\CONSTANT;
19.
20.     $obj = new namespace\Another;    // 实例化 foo\Another 对象
21.     $obj = new Another;              // 实例化 My\Full\Classname  对象
22.     NSname\subns\func();             // 调用 My\Full\NSname\subns\func 函数
23.     $a = new ArrayObject(array(1)); // 实例化 ArrayObject 对象
24.                                      // 如果不使用 "use \ArrayObject" ，则实例化一个
    foo\ArrayObject 对象
25.     func();                          // 调用 My\Full\functionName 函数
26.     echo CONSTANT;                   // 打印 My\Full\CONSTANT 常量
27. ?>
```

> 注意：对命名空间中的名称（包含命名空间分隔符的完全限定名称，如 Foo\Bar ，以及相对的不包含命名空间分隔符的全局名称，如 FooBar）来说，前导的反斜杠是不必要的也是不推荐的，因为导入的名称必须是完全限定的，不会根据当前的命名空间作相对解析。

为了简化操作，PHP 还支持在一行中导入多个命名空间，中间使用`,`隔开，示例代码如下：

```
01. <?php
02.     use My\Full\Classname as Another, My\Full\NSname;
03.
04.     $obj = new Another;      // 实例化 My\Full\Classname 对象
05.     NSname\subns\func();     // 调用 My\Full\NSname\subns\func 函数
06. ?>
```

导入操作是编译执行的，但动态的类名称、函数名称或常量名称则不是。

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

```
03.
04.     $obj = new Another; // 实例化一个 My\Full\Classname 对象
05.     $a = 'Another';
06.     $obj = new $a;      // 实际化一个 Another 对象
07. ?>
```

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。

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。

__NAMESPACE__ 常量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，它是一个空的字符串。示例代码如下：

```
01. <?php
02.     namespace App\Controller\Home;
03.     echo __NAMESPACE__;
04. ?>
```

运行结果如下：

```
App\Controller\Home
```

namespace 关键字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。示例代码如下：

```
01. <?php
02.     namespace MyProject;
03.
04.     use Foo\Bar\Demo as demo;       // 导入 Foo\Bar\Demo 命名空间
05.
06.     blah\mine();                    // 调用 MyProject\blah\mine() 函数
```

```php
    namespace\blah\mine();          // 调用 MyProject\blah\mine() 函数
    namespace\func();               // 调用 MyProject\func() 函数
    namespace\sub\func();           // 调用 MyProject\sub\func() 函数
    namespace\cname::method();      // 调用 MyProject\cname 类的静态方法 method
    $a = new namespace\sub\cname(); // 实例化 MyProject\sub\cname 对象
    $b = namespace\CONSTANT;        // 将常量 MyProject\CONSTANT 的值赋给 $b
?>
```

命名空间名称解析规则

在说明名称解析规则之前，我们先来看看命名空间名称的定义：

- 非限定名称：名称中不包含命名空间分隔符的标识符，例如 Foo；
- 限定名称：名称中含有命名空间分隔符的标识符，例如 Foo\Bar；
- 完全限定名称：名称中包含命名空间分隔符，并以命名空间分隔符开始的标识符，例如 \Foo\Bar。namespace\Foo 也是一个完全限定名称。

名称解析遵循下列规则：

- 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；
- 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()；
- 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()；
- 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()；
- 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空间中的 foo() 函数。
- 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：
  1. new C() 的解析：

- 在当前命名空间中查找 A\B\C 类；
- 尝试自动装载类 A\B\C。

2. new D\E() 的解析：
   - 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；
   - 尝试自动装载类 A\B\D\E。
3. 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()。

优秀文章

数组之间的赋值，C语言数组之间赋值详解	Linux bzip2命令：压缩文件（.bz2格式）
Java成员方法的声明和调用	C++ static静态成员函数详解
Shell expr命令：进行整数计算	URL编码是怎么回事?
Maven仓库的分类	Linux fdisk创建分区（主分区、扩展分区和逻辑分
Hibernate connection.driver_class属性——指定	netwox显示网络配置信息