php的命名空间与自动加载实现方法

作者：有故事的人 来源：范文网 www.wtabcd.cn/fanwen/

本文原地址：https://www.wtabcd.cn/fanwen/zuowen/7477b900a8c3675ed94a13b6e3baf2dd.html

类的自动加载

引子

当我们在php代码中加载类时，我们必须要include或者require 某个类文件。

但遇到类似的情况，例如:

require "class1.php";require "class2.php";$boy = $_get['sex'] = 0?true:false;if($boy){ $class1 = new class1();}else{ $class2 = new class2();}

假如我们需要判断一个人的性别，如果是男的就实例化class1这个类，如果是女的就实例化class2这个类。那么问题来了：这段代码，每次我只需要执行一个实例化对象，然而我必须加载这两个类文件。

php对于这种问题提出了解决方案

spl_auto_register()

这个概念在 在php5.1中提出

spl_auto_register($autoload_function = null, $throw = true, $prepend = false)

函数包含3个参数

①autoload_function 这是一个函数【方法】名称，可以是字符串或者数组（调用类方法使用）。这个函数（方法）的功能就是，来把需要new 的类文件包含include(requeire)进来，这样new的时候就不会找不到文件了。其实就是封装整个项目的include和require功能。

② $throw 该参数指定当autoload_function无法注册时，spl_autoload_register（）是否应引发异常世界上最小的国家4个人。

③ 如果为true，那么spl_autoload_register（）将在自动加载到文件前面，而不时在它后面。

用法

那么有了这个函数之后向这样写了

function load($class){ require "./{$class}.php";}spl_autoload_register('load');if($boy){ $class1 = new class1();}else{ $class2 = new class2();}

程序执行过程如下：

> // 正常的流程
> new 一个对象 – >找不到对象 – > 报错
>
> // 引入spl_autoload_register 后
> new 一个对象 – >找不到对象 – > spl_autoload_register对说交给我试试 – > 加载成功

加载之后我们执行了load这个函数，通过class的拼接，我们完成了加载函数的过程

__autoload()

类的自动加载在前面我们讲 spl_autoload_register 的时候已经和大家讲过了。今天我们讲另一种 __autoload() 在php7中已经不建议使用了

php的__autoload函数是一个魔术函数，在这个函数出现之前，如果一个php文件里引用了100个对象，那么这个文件就需要使用include或require引进100个类文件，这将导致该php文件无比庞大。于是就有了这个 __autoload函数。

__autoload函数在什么时候调用呢？当php文件中使用了new关键字实例化一个对象时，如果该类没有在本php文件中被定义，将会触发__autoload函数，此时，就可以引进定义该类的php文件，而后，就能实例化成功了。

> （注意：如果需要实例化的对象，在本文件中已经找到该类的定义的话，就不会触发 __autoload 函数）

他和 spl_autoload_registe
r的区别就在于当文件中同时出现__autoload和spl_autoload_register时，以spl_autoload_register为准

命名空间

我们先前讲过类的自动加载，然后我就在思索。

我们用框架写代码的时候，每在另一个文件中调用其他类时

我们并没有写spl_autoload_register这个方法啊？那我们时怎么实现的呢？

原理

原来啊，我们php在5.3时引入了命名空间的概念(这也是为什么大多数的框架不支持5.3之前的版本原因之一)，命名空间大家多少还是了解的吧：不知道的去墙角面壁思过

命名空间简而言之就是一种标识，它的主要目的是解决命名冲突的问题。就像在日常生活中，有很多姓名相同的人，如何区分这些人呢？那就需要加上一些额外的标识。把工作单位当成标识似乎不错，这样就不用担心“ 撞名 ”的尴尬了。

命名空间分类

完全限定命名空间限定命名空间

new 成都\徐大帅(); // 限定类名new \成都\徐大帅(); // 完全限定类名

在当前命名空间没有声明的情况下，限定类名和完全限定类名是等价的。因为如果不指定空间，则默认为全局（）。

namespace 美国;new 成都\徐大帅(); // 美国\成都\徐大帅（实际结果）new \成都\徐大帅();
// 成都\徐大帅（实际结果）

这个例子展示了在命名空间下，使用限定类名和完全限定类趋势的近义词名的区别。（完全限定类名 = 当前命名空间 + 限定类名）

/* 导入命名空间 */use 成都\徐大帅;new 徐大帅(); // 成都\徐大帅（实际结果）/* 设置别名 */use 成都\徐大帅 as ceo;new ceo(); // 成都\徐大帅（实际结果）/* 任何情况 */new \成都\徐大帅();// 成都\徐大帅（实际结果）

使用命名空间只是让类名有了前缀，不容易发生冲突，系统仍然不会进行自动导入。

如果不引入文件，系统会在抛出“ class not found ”错误之前触发 __autoload()
或者spl_autoload_register函数，并将限定类名传入作为参数。

上面的例子都是基于你已经将相关文件手动引入的情况下实现的，否则系统会抛出 ” class ‘ 成都徐大帅 ’ not
found ”。因为她不知道这个文件在哪里。所以在引入命名空间以后又引入了自动加载

接下来，我们就在用命名空间加载我们的 类

一个使用命名空间自动加载类的小实验

首先，我们在一个新文件中定义

//school.phpnamespace top;class school{ function __construct() { echo '这是'.__class__.'类的实现'; }}

这当然不是重要的，重要的是我们调用他的函数。我们在同一个目录建立一个index.php文件（不同文件也行，只要你写好映射关系）

```
//index.phpspl_autoload_register(function ($class){ //从我们的 class名称中找，有没有对应的路径 $map = [ 'top\\school'=>'./school.php' ]; $file = $map[$class]; //查看对应的文件是否存在 if (file_exists($file)) include $file;});echo "开始<br/>";new top\school();
```

结果

```
开始
这是top\school类的实现
```

我们使用了 类名和类地址的映射关系，实现了我们的自动加载。然而这也意味着我们每次添加文件，就必须去更新我们的映射文件。在一个大型系统中这样数组维持的映射关系无疑很麻烦。那么有没有好一点的做法呢？

psr4 自动加载规范

不知道的童鞋，可以看这里

psr4 中文文档

psr4 的具体解释

下面摘自上面链接，我觉得上面两篇文章已经讲得很透彻了

\<namespacename>(\<subnamespacenames>)*\<classname>

psr-4 规范中必须要有一个顶级命名空间，它的意义在于表示某一个特殊的目录（文件基目录）。子命名空间代表的是类文件相对于文件基目录的这一段路径（相对路径），类名则与文件名保持一致（注意大小写的区别）。

举个例子：在全限定类名 appviewnewsindex 中，如果 app 代表 c:baidu，那么这个类的路径则是 c:baiduviewnewsindex.php

我们就以解析 appviewnewsindex 为例，编写一个简单的 demo：

```
$class = 'app\view\news\index';/* 顶级命名空间路径映射 */$vendor_map = array( 'app' => 'c:\baidu',);/* 解析类名为文件路径 */$vendor = substr($class, 0, strpos($class, '\\')); // 取出顶级命名空间[app]$vendor_dir = $vendor_map[$vendor]; // 文件基目录[c:\baidu] $rel_path = dirname(substr($class, strlen($vendor))); // 相对路径[/view/news]$file_name = basename($class) . '.php'; // 文件名[index.php]/* 输出文件所在路径 */echo $vendor_dir . $rel_path . directory_separator . $file_name;
```

通过这个 demo

可以看出限定类名转换为路径的过程。那么现在就让我们用规范的面向酥肉怎么做好吃做法步骤对象方式去实现自动加载器吧。

首先我们创建一个文件 index.php，它处于 appmvcviewhome 目录中：

```
namespace app\mvc\view\home;class index{ function __construct() { echo '<h1> welco
me to home </h1>'; }}
```

接着我们在创建一个加载类（不需要命名空间），它处于 目录中：

```
class loader{ /* 路径映射 */ public static $vendormap = array( 'app' => __dir__
 . directory_separator . 'app', ); /** * 自动中国高考网加载器 */ public static f
unction autoload($class) { $file = self::findfile($class); if (file_exists($file)) { self:
:includefile($file); } } /** * 解析文件路径 */ private static function findfile($clas
s) { $vendor = substr($class, 0, strpos($class, '\\')); // 顶级命名空间 $vendordir
 = self::$vendormap[$vendor]; // 文件基目录 $filepath = substr($class, strlen($vendor))
 . '.php'; // 计算机二级操作题文件相对路径 return strtr($vendordir . $filepath, '\\',
 directory_separator); // 文件标准路径 } /** * 引入文件 */ private static functi
on includefile($file) { if (is_file($file)) { include $file; } }}
```

最后，将 loader 类中的 autoload 注册到 spl_autoload_register 函数中：

```
include 'loader.php'; // 引入加载器spl_autoload_register('loader::autoload'); // 注册自动加载n
ew \app\mvc\view\home\index(); // 实例化未引用的类/** * 输出: <h1> welcome to ho
me </h1> */
```

示例中的代码其实就是 thinkphp 自动加载器源码的精简版，它是 thinkphp 5 能实现惰性加载的关键。

总结

以上就是这篇文章的全部内容了，希望本文的内容对大家的学习或者工作具有一定的参考学习价值，谢谢大家对www.887551.com的支持。