# PSR-4 相关示例

以下是 PSR-4 规范的相关示例代码：

## 闭包的实现示例

<?php  
/\*\*  
 \* 一个具体项目的实例  
 \*   
 \* 当使用 SPL 注册此自动加载器后，执行以下语句将从   
 \* /path/to/project/src/Baz/Qux.php 载入 \Foo\Bar\Baz\Qux 类：  
 \*   
 \* new \Foo\Bar\Baz\Qux;  
 \*   
 \* @param string $class 完整的类名  
 \* @return void  
 \*/  
spl\_autoload\_register(function ($class) {  
   
 // 具体项目命名空间前缀  
 $prefix = 'Foo\\Bar\\';  
  
 // 命名空间前缀的基目录  
 $base\_dir = \_\_DIR\_\_ . '/src/';  
   
 // 判断类名是否具有本命名空间前缀  
 $len = strlen($prefix);  
 if (strncmp($prefix, $class, $len) !== 0) {  
 // 不含本命名空间前缀，退出本自动载入器  
 return;  
 }  
   
 // 截取相应类名  
 $relative\_class = substr($class, $len);  
   
 // 将命名空间前缀替作为文件基目录，然后  
 // 将类名中的命名空间分隔符替换成文件分隔符,  
 // 最后添加 .php 后缀  
 $file = $base\_dir . str\_replace('\\', '/', $relative\_class) . '.php';  
   
 // 如果以上文件存在，则将其载入  
 if (file\_exists($file)) {  
 require $file;  
 }  
});

## 类的实现示例

下面是一个可处理多命名空间的类实例

<?php  
namespace Example;  
  
/\*\*  
 \* An example of a general-purpose implementation that includes the optional  
 \* functionality of allowing multiple base directories for a single namespace  
 \* prefix.  
 \*   
 \* Given a foo-bar package of classes in the file system at the following  
 \* paths ...  
 \*   
 \* /path/to/packages/foo-bar/  
 \* src/  
 \* Baz.php # Foo\Bar\Baz  
 \* Qux/  
 \* Quux.php # Foo\Bar\Qux\Quux  
 \* tests/  
 \* BazTest.php # Foo\Bar\BazTest  
 \* Qux/  
 \* QuuxTest.php # Foo\Bar\Qux\QuuxTest  
 \*   
 \* ... add the path to the class files for the \Foo\Bar\ namespace prefix  
 \* as follows:  
 \*   
 \* <?php  
 \* // instantiate the loader  
 \* $loader = new \Example\Psr4AutoloaderClass;  
 \*   
 \* // register the autoloader  
 \* $loader->register();  
 \*   
 \* // register the base directories for the namespace prefix  
 \* $loader->addNamespace('Foo\Bar', '/path/to/packages/foo-bar/src');  
 \* $loader->addNamespace('Foo\Bar', '/path/to/packages/foo-bar/tests');  
 \*   
 \* The following line would cause the autoloader to attempt to load the  
 \* \Foo\Bar\Qux\Quux class from /path/to/packages/foo-bar/src/Qux/Quux.php:  
 \*   
 \* <?php  
 \* new \Foo\Bar\Qux\Quux;  
 \*   
 \* 以下代码将由 /path/to/packages/foo-bar/tests/Qux/QuuxTest.php   
 \* 载入 \Foo\Bar\Qux\QuuxTest 类  
 \*   
 \* <?php  
 \* new \Foo\Bar\Qux\QuuxTest;  
 \*/  
class Psr4AutoloaderClass  
{  
 /\*\*  
 \* An associative array where the key is a namespace prefix and the value  
 \* is an array of base directories for classes in that namespace.  
 \*  
 \* @var array  
 \*/  
 protected $prefixes = array();  
  
 /\*\*  
 \* 在 SPL 自动加载器栈中注册加载器  
 \*   
 \* @return void  
 \*/  
 public function register()  
 {  
 spl\_autoload\_register(array($this, 'loadClass'));  
 }  
  
 /\*\*  
 \* 添加命名空间前缀与文件基目录对  
 \*  
 \* @param string $prefix 命名空间前缀  
 \* @param string $base\_dir 命名空间中类文件的基目录  
 \* @param bool $prepend 为 True 时，将基目录插到最前，这将让其作为第一个被搜索到，否则插到将最后。  
 \* @return void  
 \*/  
 public function addNamespace($prefix, $base\_dir, $prepend = false)  
 {  
 // 规范化命名空间前缀  
 $prefix = trim($prefix, '\\') . '\\';  
   
 // 规范化文件基目录  
 $base\_dir = rtrim($base\_dir, '/') . DIRECTORY\_SEPARATOR;  
 $base\_dir = rtrim($base\_dir, DIRECTORY\_SEPARATOR) . '/';  
  
 // 初始化命名空间前缀数组  
 if (isset($this->prefixes[$prefix]) === false) {  
 $this->prefixes[$prefix] = array();  
 }  
   
 // 将命名空间前缀与文件基目录对插入保存数组  
 if ($prepend) {  
 array\_unshift($this->prefixes[$prefix], $base\_dir);  
 } else {  
 array\_push($this->prefixes[$prefix], $base\_dir);  
 }  
 }  
  
 /\*\*  
 \* 由类名载入相应类文件  
 \*  
 \* @param string $class 完整的类名  
 \* @return mixed 成功载入则返回载入的文件名，否则返回布尔 false  
 \*/  
 public function loadClass($class)  
 {  
 // 当前命名空间前缀  
 $prefix = $class;  
   
 // work backwards through the namespace names of the fully-qualified  
 // class name to find a mapped file name  
 while (false !== $pos = strrpos($prefix, '\\')) {  
   
 // retain the trailing namespace separator in the prefix  
 $prefix = substr($class, 0, $pos + 1);  
  
 // the rest is the relative class name  
 $relative\_class = substr($class, $pos + 1);  
  
 // try to load a mapped file for the prefix and relative class  
 $mapped\_file = $this->loadMappedFile($prefix, $relative\_class);  
 if ($mapped\_file) {  
 return $mapped\_file;  
 }  
  
 // remove the trailing namespace separator for the next iteration  
 // of strrpos()  
 $prefix = rtrim($prefix, '\\');   
 }  
   
 // 找不到相应文件  
 return false;  
 }  
   
 /\*\*  
 \* Load the mapped file for a namespace prefix and relative class.  
 \*   
 \* @param string $prefix The namespace prefix.  
 \* @param string $relative\_class The relative class name.  
 \* @return mixed Boolean false if no mapped file can be loaded, or the  
 \* name of the mapped file that was loaded.  
 \*/  
 protected function loadMappedFile($prefix, $relative\_class)  
 {  
 // are there any base directories for this namespace prefix?  
 if (isset($this->prefixes[$prefix]) === false) {  
 return false;  
 }  
   
 // look through base directories for this namespace prefix  
 foreach ($this->prefixes[$prefix] as $base\_dir) {  
  
 // replace the namespace prefix with the base directory,  
 // replace namespace separators with directory separators  
 // in the relative class name, append with .php  
 $file = $base\_dir  
 . str\_replace('\\', DIRECTORY\_SEPARATOR, $relative\_class)  
 . '.php';  
 $file = $base\_dir  
 . str\_replace('\\', '/', $relative\_class)  
 . '.php';  
  
 // 当文件存在时，在入之  
 if ($this->requireFile($file)) {  
 // 完成载入  
 return $file;  
 }  
 }  
   
 // 找不到相应文件  
 return false;  
 }  
   
 /\*\*  
 \* 当文件存在，则从文件系统载入之  
 \*   
 \* @param string $file 需要载入的文件  
 \* @return bool 当文件存在则为 True，否则为 false  
 \*/  
 protected function requireFile($file)  
 {  
 if (file\_exists($file)) {  
 require $file;  
 return true;  
 }  
 return false;  
 }  
}

### 单元测试

以下是上面代码单元测试的一种实现：

```php <?php namespace Example;

class MockPsr4AutoloaderClass extends Psr4AutoloaderClass { protected $files = array();

public function setFiles(array $files)  
{  
 $this->files = $files;  
}  
  
protected function requireFile($file)  
{  
 return in\_array($file, $this->files);  
}

}

class Psr4AutoloaderClassTest extends \_Framework\_TestCase { protected $loader;

protected function setUp()  
{  
 $this->loader = new MockPsr4AutoloaderClass;  
  
 $this->loader->setFiles(array(  
 '/vendor/foo.bar/src/ClassName.php',  
 '/vendor/foo.bar/src/DoomClassName.php',  
 '/vendor/foo.bar/tests/ClassNameTest.php',  
 '/vendor/foo.bardoom/src/ClassName.php',  
 '/vendor/foo.bar.baz.dib/src/ClassName.php',  
 '/vendor/foo.bar.baz.dib.zim.gir/src/ClassName.php',  
 ));  
   
 $this->loader->addNamespace(  
 'Foo\Bar',  
 '/vendor/foo.bar/src'  
 );  
   
 $this->loader->addNamespace(  
 'Foo\Bar',  
 '/vendor/foo.bar/tests'  
 );  
   
 $this->loader->addNamespace(  
 'Foo\BarDoom',  
 '/vendor/foo.bardoom/src'  
 );  
   
 $this->loader->addNamespace(  
 'Foo\Bar\Baz\Dib',  
 '/vendor/foo.bar.baz.dib/src'  
 );  
   
 $this->loader->addNamespace(  
 'Foo\Bar\Baz\Dib\Zim\Gir',  
 '/vendor/foo.bar.baz.dib.zim.gir/src'  
 );  
}  
  
public function testExistingFile()  
{  
 $actual = $this->loader->loadClass('Foo\Bar\ClassName');  
 $expect = '/vendor/foo.bar/src/ClassName.php';  
 $this->assertSame($expect, $actual);  
   
 $actual = $this->loader->loadClass('Foo\Bar\ClassNameTest');  
 $expect = '/vendor/foo.bar/tests/ClassNameTest.php';  
 $this->assertSame($expect, $actual);  
}  
  
public function testMissingFile()  
{  
 $actual = $this->loader->loadClass('No\_Vendor\No\_Package\NoClass');  
 $this->assertFalse($actual);  
}  
  
public function testDeepFile()  
{  
 $actual = $this->loader->loadClass('Foo\Bar\Baz\Dib\Zim\Gir\ClassName');  
 $expect = '/vendor/foo.bar.baz.dib.zim.gir/src/ClassName.php';  
 $this->assertSame($expect, $actual);  
}  
  
public function testConfusion()  
{  
 $actual = $this->loader->loadClass('Foo\Bar\DoomClassName');  
 $expect = '/vendor/foo.bar/src/DoomClassName.php';  
 $this->assertSame($expect, $actual);  
   
 $actual = $this->loader->loadClass('Foo\BarDoom\ClassName');  
 $expect = '/vendor/foo.bardoom/src/ClassName.php';  
 $this->assertSame($expect, $actual);  
}

}