





- 
- 



在开发过程中，我们经常会遇到需要测试一些外部依赖或服务的情况。这时候，我们就需要使用 Mock 技术来模拟这些依赖或服务，以便在不依赖真实环境的情况下进行单元测试。

# 1. 什么是 Mock

Mock 是一种编程技术，用于模拟对象的行为。在 PHP 中，我们通常使用 PHPUnit 和 Mockery 来实现 Mock。

在 PHP 中，Mock 通常用于测试类的方法。通过 Mock，我们可以模拟一个类的行为，而不需要创建真实的对象。

例如，我们可以使用 Mockery 来模拟一个接口或抽象类的方法。

```
namespace App\Services;

interface GreetingServiceInterface
{
    public function greet($name);
}

class GreetingService implements GreetingServiceInterface
{
    public function greet($name)
    {
        return "Hello, $name!";
    }
}
```

在测试中，我们可以使用 Mockery 来模拟 GreetingServiceInterface 的方法。

```
namespace App\Http\Controllers;

use App\Services\GreetingServiceInterface;

class GreetingController extends Controller
{
    protected $greetingService;
```

```

public function __construct(GreetingServiceInterface $greetingService)
{
    $this->greetingService = $greetingService;
}

public function greet($name)
{
    return $this->greetingService->greet($name);
}
}

```

이제 이 코드를 테스트해 보겠습니다.

```

use Tests\TestCase;
use App\Services\GreetingServiceInterface;
use App\Http\Controllers\GreetingController;
use Mockery;

class GreetingControllerTest extends TestCase
{
    public function testGreet()
    {
        // GreetingServiceInterface를 모킹합니다
        $mockService = Mockery::mock(GreetingServiceInterface::class);

        // greet 메서드가 "Hello, John!"을 반환하도록 모킹합니다
        $mockService->shouldReceive('greet')
            ->with('John')
            ->andReturn('Hello, John!');

        // 컨트롤러를 생성합니다
        $controller = new GreetingController($mockService);

        // 테스트 실행
        $response = $controller->greet('John');
        $this->assertEquals('Hello, John!', $response);
    }

    protected function tearDown(): void
    {

```

```
Mockery::close();
parent::tearDown();
}
}
```

## 2. 创建测试用例

首先，我们需要创建一个测试用例类，用于测试我们的服务提供者。

在 `tests` 目录下，

创建 `GreetingServiceTest.php` 文件。

```
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use App\Services\GreetingServiceInterface;
use App\Services\GreetingService;

class AppServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->bind(GreetingServiceInterface::class, GreetingService::class);
    }

    public function boot()
    {
        //
    }
}
```

在 `tests` 目录下，

```
use Tests\TestCase;
use App\Services\GreetingServiceInterface;

class GreetingServiceTest extends TestCase
{
```

```

public function testGreet()
{
    // 创建 GreetingService 的实例
    $greetingService = $this->app->make(GreetingServiceInterface::class);

    // 调用 greet 方法
    $response = $greetingService->greet('John');
    $this->assertEquals('Hello, John!', $response);
}
}

```



- **Mockery** 是一个强大的 PHP 测试框架，用于创建测试用的模拟对象。它支持多种测试框架，如 PHPUnit、Symfony 等。
- **PHPUnit** 是一个流行的 PHP 单元测试框架，它提供了丰富的断言方法和测试用例支持。

**Q1:** Mock 对象在测试中有什么作用？

**Q2:** 如何创建一个 Mock 对象？

**Q3:** 在测试中，如何验证 Mock 对象的行为？

You wanna more detailed information?



<https://darkghosthunter.medium.com/php-10-tips-to-use-for-mockery-33673ba01321>

- 1. 如何设置Mockery
- 2. 如何设置Mockery
- 3. 如何设置Mockery, Mockery的Mockery::mock()方法, expects()方法, allows()方法

Mockery是一个PHP的Mocking框架, 它可以帮助你测试你的代码, 而不需要依赖真实的数据库或网络服务。

```
$chain = Mockery::mock(Chainable::class);

$chain->expects('foo')->andReturnSelf();
$chain->expects('bar')->with(true)->andReturnSelf();
$chain->expects('baz')->with(10)->andReturnSelf();
```

Mockery的Mockery::mock()方法, 它可以帮助你测试你的代码, 而不需要依赖真实的数据库或网络服务。

```
$mock = Mockery::mock(Chainable::class);

$mock->expects('foo->bar->baz')->andReturn('foo');
```

Mockery的Mockery::mock()方法, 它可以帮助你测试你的代码, 而不需要依赖真实的数据库或网络服务。

```
$mock = Mockery::mock(MyObject::class);

$mock->expects()->first()->ordered();
$mock->expects()->second()->ordered();
```

. 順序を指定して期待値を設定する場合は、`ordered()` を呼び出す必要がある。この場合、期待値は順番に検証される。

期待値を `ordered()` で指定すると、期待値は順番に検証される。

## 9. 期待値を指定する

期待値を指定する場合は、`Mockery::mock()` を呼び出す必要がある。

```
$mock = Mockery::spy(Useful::class);

// An object uses the mock.

$mock->shouldHaveReceived('getBody')->times(3);
```

期待値を指定する場合は、`Mockery::spy()` を呼び出す必要がある。この場合、期待値は順番に検証される。

期待値を指定する場合は、`Mockery::mock()` を呼び出す必要がある。

```
$mock = Mockery::mock(Useful::class);

$mock->expects('foo')
    ->withAny()
    ->andReturnUsing(fn ($string) => 'You wrote: ' . $string)
```