79. 中间件.上

学习要点：

1.定义中间件
2.前/后置
3.结束调度

本节课我们来学习一下中间件的用法，定义一下中间件。

一. 定义中间件

1. 中间件的主要用于拦截和过滤 HTTP 请求，并进行相应处理；
2. 这些请求的功能可以是 URL 重定向、权限验证等等；
3. 为了进一步了解中间件的用法，我们首先定义一个基础的中间件；
4. 可以通过命令行模式，在应用目录下生成一个中间件文件和文件夹；

```
php think make:middleware Check
```

```
namespace app\middleware;
class Check
{
    public function handle($request, \Closure $next)
    {
        if ($request->param('name') == 'index') {
            return redirect('../');
        }
        return $next($request);
    }
}
```

5. 然后将这个中间件进行注册，在应用目录下创建 middleware.php 中间件配置；

```
return [
    app\middleware\Check::class
];
```

6. 中间件的入口执行方法必须是：handle()方法，第一参数请求，第二参数是闭包；
7. 业务代码判断请求的 name 如果等于 index，就拦截住，执行中间件，跳转到首页；
8. 但如果请求的 name 是 lee，那需要继续往下执行才行，不能被拦死；
9. 那么就需要$next($request)把这个请求去调用回调函数；
10. 中间件 handle()方法规定需要返回 response 对象，才能正常使用；
11. 而$next($request)执行后，就是返回的 response 对象；
12. 为了测试拦截后，无法继续执行，可以 return response()助手函数测试；

二．前/后置中间件

1. 将$next($request)放在方法底部的方式，属于前置中间件；
2. 前置中间件就是请求阶段来进行拦截验证，比如登录判断、跳转、权限等；
3. 而后置中间件就是请求完毕之后再进行验证，比如写入日志等等；

```php
public function handle($request, \Closure $next)
{
    //中间件代码，前置
    return $next($request);
}

public function handle($request, \Closure $next)
{
    $response = $next($request);
    //中间件代码，后置
    return $response;
}
```

三．结束调度

1. 中间件提供了一个 end()方法，可以在中间件执行到最后时执行；

```php
public function end(Response $response)
{
    //中间件执行到最后执行
    echo $response->getData();
}
```