PHP 设计模式 - 状态模式

作者：henryspace
原文链接：https://ld246.com/article/1590400117406
来源网站：链滴

状态模式当一个对象的内在状态改变时允许改变其行为，这个对象看起来像是改变了其类。状态模式要解决的是当控制一个对象状态的条件表达式过于复杂时的情况。把状态的判断逻辑转移到表示不同态的一系列类中，可以把复杂的判断逻辑简化。

角色

- 上下文环境（Context）：它定义了客户程序需要的接口并维护一个具体状态角色的实例，将与状相关的操作委托给当前的具体对象来处理。
- 抽象状态（State）：定义一个接口以封装使用上下文环境的的一个特定状态相关的行为。
- 具体状态（StateObj）：具体实现抽象状态定义的接口。

实现代码

```
/**
 * 状态接口
 * Interface State
 */
interface State {

    public function Handle(Context $context);

}

/**
 * 状态上下文环境
 * Class Context
 */
class Context {

    private  $_state;
    public function __construct(State $state)
    {
        $this->_state = $state;
    }

    public function setState(State $state)
    {
        $this->_state = $state;
    }

    public function handle() {

        return $this->_state->handle($this);
    }

}

/**
 * 订单状态处理
 * Class Order
 */
```

```
class Order implements State {

    //定义常量，订单的五种状态
    const ORDER_STATUS_NEW = 'new';
    const ORDER_STATUS_PAYED = 'payed';
    const ORDER_STATUS_DELIVERY = 'delivery';
    const ORDER_STATUS_RECEIVED = 'received';
    const ORDER_STATUS_COMPLETE = 'complete';

    public $status; //当前状态

    /**
     * 统一状态变更管理
     * @param Context $context
     * @return mixed
     */
    public function handle(Context $context) {

        return $this->dispatch($context);
    }

    /**
     * 状态分发处理
     * @param Context $context
     * @return array
     */
    public function dispatch(Context $context) {

        switch($this->status) {

            case self::ORDER_STATUS_NEW:
                $result = $this->handleForNew();
                break;
            case self::ORDER_STATUS_PAYED:
                $result = $this->handleForPayed();
                break;
            case self::ORDER_STATUS_DELIVERY:
                $result = $this->handleForDelivery();
                break;
            case self::ORDER_STATUS_RECEIVED:
                $result = $this->handleForReceived();
                break;
            case self::ORDER_STATUS_COMPLETE:
                $result = $this->handleForComplete();
                break;
            default:;
        }

        return $result;
    }
    public function handleForNew() {

        //TODO
    }
```

```
    public function handleForPayed() {

        //TODO
    }
    public function handleForDelivery() {

        //TODO
    }
    public function handleForComplete() {

        //TODO
    }

}
```

具体使用

```
# 可避免controller层耦合太多逻辑和状态判断

$state = new Order();
$state->status = $state::ORDER_STATUS_NEW;

$context = new Context($state);
$result = $context->handle();
```