登录 | 注册

guisu，程序人生。 逆水行舟，不进则退。

能干的人解决问题。智慧的人绕开问题(A clever person solves a problem. A wise person avoids it)

目录视图 摘要视图 订阅

友情链接

诚聘：
http://job.csdn.net/job/index?jobID=81336

个人资料

真实的归宿

访问： 2365999次
积分： 20193
等级：
排名： 第168名

原创： 206篇 转载： 2篇
译文： 0篇 评论： 839条

文章分类

操作系统 (5)
Linux (21)
MySQL (12)
PHP (42)
PHP内核 (11)
技术人生 (7)
数据结构与算法 (30)
云计算hadoop (25)
网络知识 (7)
c/c++ (23)
memcache (5)
HipHop (2)
计算机原理 (4)
Java (7)
socket网络编程 (8)
设计模式 (26)
AOP (2)
重构 (11)
重构与模式 (1)
大数据处理 (12)
搜索引擎Search Engine (15)
HTML5 (1)
Android (1)
webserver (3)
NOSQL (7)

CSDN博乐 举荐之美 公益活动，感谢你们 3D游戏引擎实战班4个月只需1999元！ 新版极客头条上线，每天一大波干货

设计模式 (十七) 状态模式State（对象行为型）

分类： 设计模式 2012-05-11 17:26 24900人阅读 评论(17) 收藏 举报

设计模式 function lift class 扩展 手机

设计模式 (十七) 状态模式State（对象行为型）

1.概述

在软件开发过程中，应用程序可能会根据不同的情况作出不同的处理。最直接的解决方案是将这些所有可能发生的情况全都考虑到。然后使用if... ellse语句来做状态判断来进行不同情况的处理。但是对复杂状态的判断就显得“力不从心了”。随着增加新的状态或者修改一个状体（if else(或switch case)语句的增多或者修改）可能会引起很大的修改，而程序的可读性，扩展性也会变得很弱。维护也会很麻烦。那么我就考虑只修改自身状态的模式。

例子1：按钮来控制一个电梯的状态，一个电梯开们，关门，停，运行。每一种状态改变，都有可能要根据其他状态来更新处理。例如，开门状体，你不能在运行的时候开门，而是在电梯定下后才能开门。

例子2：我们给一部手机打电话，就可能出现这几种情况：用户开机，用户关机，用户欠费停机，用户消户等。 所以当我们拨打这个号码的时候：系统就要判断，该用户是否在开机且不忙状态，又或者是关机，欠费等状态。但不管是那种状态我们都应给出对应的处理操作。

2.问题

对象如何在每一种状态下表现出不同的行为？

3.解决方案

状态模式：允许一个对象在其内部状态改变时改变它的行为。对象看起来似乎修改了它的类。

在很多情况下，一个对象的行为取决于一个或多个动态变化的属性，这样的属性叫做状态，这样的对象叫做有状态的(stateful)对象，这样的对象状态是从事先定义好的一系列值中取出的。当一个这样的对象与外部事件产生互动时，其内部状态就会改变，从而使得系统的行为也随之发生变化。

4.适用性

在下面的两种情况下均可使用State模式:

1) • 一个对象的行为取决于它的状态, 并且它必须在运行时刻根据状态改变它的行为。

2) • 代码中包含大量与对象状态有关的条件语句:一个操作中含有庞大的多分支的条件（if else(或switch case)语句，且这些分支依赖于该对象的状态。这个状态通常用一个或多个枚举常量表示。通常，有多个操作包含这一相同的条件结构。 State模式将每一个条件分支放入一个独立的类中。这使得你可以根据对象自身的情况将对象的状态作为一个对象，这一对象可以不依赖于其他对象而独立变化。

5.结构

NOSQL Mongo (0)
分布式 (1)
数据结构与算法 xi (0)
协议 (1)
信息论的熵 (0)
关于php的libevent扩展的应用 (0)
libevent简单介绍 (0)
SOA (0)

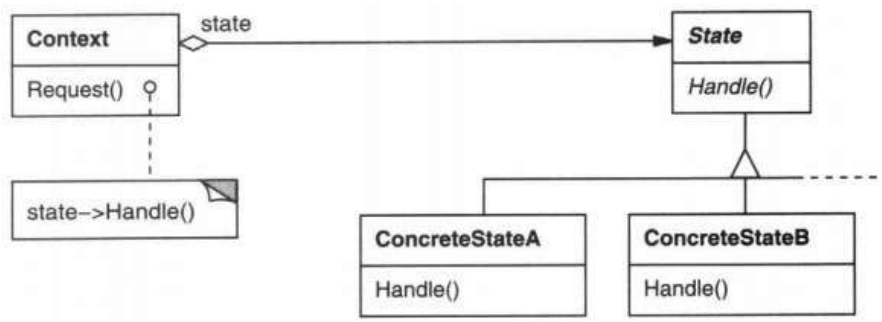
文章存档
2015年05月 (2)
2015年04月 (1)
2015年01月 (2)
2014年10月 (1)
2014年09月 (1)
展开

阅读排行
八大排序算法 (149047)
Mysql 多表联合查询效率 (115534)
socket阻塞与非阻塞，同 (76928)
hbase安装配置（整合到 (74219)
Nginx工作原理和优化、i (66084)
深入理解java异常处理机 (58399)
设计模式 (十八) 策略模 (55133)
Hadoop集群配置（最全 (54576)
Java输入输出流 (50884)
PageRank算法 (46558)

评论排行
八大排序算法 (54)
深入理解java异常处理机 (48)
socket阻塞与非阻塞，同 (46)
硬盘的读写原理 (35)
设计模式 (十八) 策略模 (35)
海量数据处理算法—Bit-M (25)
设计模式（一）工厂模式 (24)
UML图中类之间的关系:依 (23)
PHP SOCKET编程 (22)
HDFS写入和读取流程 (20)

推荐文章
* Android HandlerThread 源码分析
*从零学编程：写一封情书
*小胖学PHP总结：PHP的循环语句
*Spring基于注解@AspectJ的AOP
*HTML5梦幻之旅：仿Qt示例Drag and Drop Robot（换装机器人）

最新评论
Hadoop Hive sql语法详解
lifangzhen0: hiveSQL入门，写



6.模式的组成

环境类（Context）：定义客户感兴趣的接口。维护一个ConcreteState子类的实例，这个实例定义当前状态。

抽象状态类（State）：定义一个接口以封装与Context的一个特定状态相关的行为。

具体状态类（ConcreteState）：每一子类实现一个与Context的一个状态相关的行为。

7.效果

State模式有下面一些效果:

状态模式的优点：

1) 它将与特定状态相关的行为局部化，并且将不同状态的行为分割开来: State模式将所有与一个特定的状态相关的行为都放入一个对象中。因为所有与状态相关的代码都存在于某一个State子类中, 所以通过定义新的子类可以很容易的增加新的状态和转换。另一个方法是使用数据值定义内部状态并且让 Context操作来显式地检查这些数据。但这样将会使整个Context的实现中遍布看起来很相似的条件if else语句或switch case语句。增加一个新的状态可能需要改变若干个操作, 这就使得维护变得复杂了。State模式避免了这个问题, 但可能会引入另一个问题, 因为该模式将不同状态的行为分布在多个State子类中。这就增加了子类的数目，相对于单个类的实现来说不够紧凑。但是如果有许多状态时这样的分布实际上更好一些, 否则需要使用巨大的条件语句。正如很长的过程一样，巨大的条件语句是不受欢迎的。它们形成一大整块并且使得代码不够清晰，这又使得它们难以修改和扩展。 State模式提供了一个更好的方法来组织与特定状态相关的代码。决定状态转移的逻辑不在单块的 i f或s w i t c h语句中, 而是分布在State子类之间。将每一个状态转换和动作封装到一个类中，就把着眼点从执行状态提高到整个对象的状态。这将使代码结构化并使其意图更加清晰。

2) 它使得状态转换显式化: 当一个对象仅以内部数据值来定义当前状态时，其状态仅表现为对一些变量的赋值，这不够明确。为不同的状态引入独立的对象使得转换变得更加明确。而且, State对象可保证Context不会发生内部状态不一致的情况，因为从 Context的角度看，状态转换是原子的—只需重新绑定一个变量(即Context的State对象变量)，而无需为多个变量赋值

3) State对象可被共享 如果State对象没有实例变量—即它们表示的状态完全以它们的类型来编码—那么各Context对象可以共享一个State对象。当状态以这种方式被共享时, 它们必然是没有内部状态, 只有行为的轻量级对象。

状态模式的缺点：

1) 状态模式的使用必然会增加系统类和对象的个数。

2) 状态模式的结构与实现都较为复杂，如果使用不当将导致程序结构和代码的混乱。

8.实现

我们用电梯的例子来说明：

简单地实现代码：

```php
<?php
abstract class ILift {
    //电梯的四个状态
    const OPENING_STATE = 1;  //门敞状态
    const CLOSING_STATE = 2;  //门闭状态
    const RUNNING_STATE = 3;  //运行状态
    const STOPPING_STATE = 4; //停止状态;


    //设置电梯的状态
    public abstract function setState($state);
```

的很好

八大排序算法
雅阁驸马: @iaction:对，应该这样写，楼主那样写奇数个数的时候找到的不是最后一个非叶子节点

Linux的SOCKET编程详解
K346K346: 太好了，此乃佳作，怒赞！循序渐进，受益匪浅！

操作系统内存管理——分区、页:
hnyz0746: 很好

八大排序算法
nihaoljq2010: 感谢楼主分享！我将每个算法运行了一遍，其中nlogn算法用亿级数量级；已全部实现。中间一些算法有误，...

Linux 内存管理
matlab_fft: 赞！

socket阻塞与非阻塞，同步与异
matlab_fft: 写的真的很赞，有种相见恨晚的赶脚，楼主辛苦啦！！！！

深入理解java异常处理机制
FreMaN2011: 写的太好了!!!看了一下就懂了!谢谢楼主!!

海量数据处理算法—Bit-Map
Algoogle: @chenxicigema:是啊，不仔细看。还吓一跳呢

Java输入输出流
zongmumask: 太详细了，谢谢

友情链接

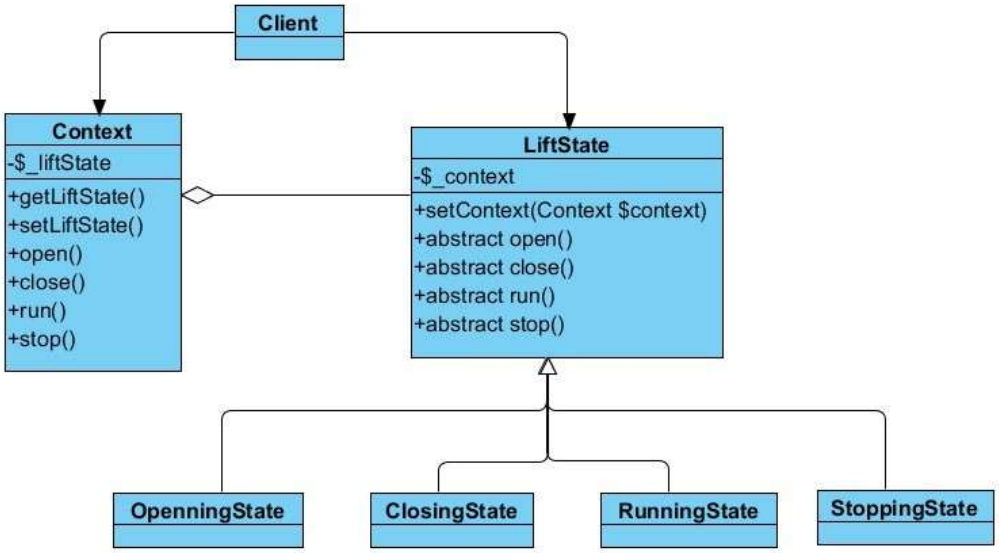
图灵机器人：聊天api的最佳选择

```
    //首先电梯门开启动作
    public abstract function open();

    //电梯门有开启，那当然也就有关闭了
    public abstract function close();

    //电梯要能上能下，跑起来
    public abstract function run();

    //电梯还要能停下来，停不下来那就扯淡了
    public abstract function stop();
}

/**
 * 电梯的实现类
 */
class Lift extends  ILift {
    private $state;

    public function setState($state) {
        $this->state = $state;
    }
    //电梯门关闭
    public function close() {
        //电梯在什么状态下才能关闭
        switch($this->state){
            case ILift::OPENING_STATE:  //如果是则可以关门，同时修改电梯状态
                $this->setState(ILift::CLOSING_STATE);
            break;
            case ILift::CLOSING_STATE:  //如果电梯就是关门状态，则什么都不做
                //do nothing;
                return ;
            break;
            case ILift::RUNNING_STATE: //如果是正在运行，门本来就是关闭的，也说明都不做
                //do nothing;
                return ;
            break;
            case ILift::STOPPING_STATE:  //如果是停止状态，本也是关闭的，什么也不做
                //do nothing;
                return ;
            break;
        }
                echo 'Lift colse <br>';
    }

    //电梯门开启
    public function open() {
        //电梯在什么状态才能开启
        switch($this->state){
            case ILift::OPENING_STATE: //如果已经在门敞状态，则什么都不做
                //do nothing;
                return ;
            break;
            case ILift::CLOSING_STATE: //如是电梯时关闭状态，则可以开启
                $this->setState(ILift::OPENING_STATE);
            break;
            case ILift::RUNNING_STATE: //正在运行状态，则不能开门，什么都不做
            //do nothing;
                return ;
            break;
            case ILift::STOPPING_STATE: //停止状态，淡然要开门了
                $this->setState(ILift::OPENING_STATE);
            break;
        }
        echo 'Lift open <br>';
    }
    ///电梯开始跑起来
    public function run() {
        switch($this->state){
            case ILift::OPENING_STATE: //如果已经在门敞状态，则不你能运行，什么都不做
                //do nothing;
                return ;
            break;
            case ILift::CLOSING_STATE: //如是电梯时关闭状态，则可以运行
                $this->setState(ILift::RUNNING_STATE);
            break;
            case ILift::RUNNING_STATE: //正在运行状态，则什么都不做
                //do nothing;
                return ;
            break;
            case ILift::STOPPING_STATE: //停止状态，可以运行
                $this->setState(ILift::RUNNING_STATE);
        }
        echo 'Lift run <br>';
    }

    //电梯停止
    public function stop() {
        switch($this->state){
            case ILift::OPENING_STATE: //如果已经在门敞状态，那肯定要先停下来的，什么都不做
                //do nothing;
                return ;
            break;
            case ILift::CLOSING_STATE: //如是电梯时关闭状态，则当然可以停止了
                $this->setState(ILift::CLOSING_STATE);
            break;
            case ILift::RUNNING_STATE: //正在运行状态，有运行当然那也就有停止了
                $this->setState(ILift::CLOSING_STATE);
            break;
            case ILift::STOPPING_STATE: //停止状态，什么都不做
                //do nothing;
                return ;
            break;
        }
        echo 'Lift stop <br>';
    }

}
```

```php
$lift = new Lift();

//电梯的初始条件应该是停止状态
$lift->setState(ILift::STOPPING_STATE);
//首先是电梯门开启，人进去
$lift->open();

//然后电梯门关闭
$lift->close();

//再然后，电梯跑起来，向上或者向下
$lift->run();
 //最后到达目的地，电梯挺下来
$lift->stop();
```

显然我们已经完成了我们的基本业务操作，但是，我们在程序中使用了大量的switch...case这样的判断（if...else也是一样),首先是程序的可阅读性很差，其次扩展非常不方便。一旦我们有新的状态加入的话，例如新加通电和断点状态。我们势必要在每个业务方法里边增加相应的case语句。也就是四个函数open，close，run，stop都需要修改相应case语句。

状态模式：把不同状态的操作分散到不同的状态对象里去完成。看看状态类的uml类图：



代码实现：

```php
<?php
/**
 *
 * 定义一个电梯的接口
 */
abstract class LiftState{

    //定义一个环境角色，也就是封装状态的变换引起的功能变化
    protected  $_context;

    public function setContext(Context $context){
        $this->_context = $context;
    }

    //首先电梯门开启动作
    public abstract function open();

    //电梯门有开启，那当然也就有关闭了
    public abstract function close();

    //电梯要能上能下，跑起来
    public abstract function run();

    //电梯还要能停下来，停不下来那就扯淡了
    public abstract function stop();

}


/**
 * 环境类:定义客户感兴趣的接口。维护一个ConcreteState子类的实例，这个实例定义当前状态。
 */
class Context {
    //定义出所有的电梯状态
    static  $openningState = null;
    static  $closeingState = null;
    static  $runningState  = null;
    static  $stoppingState = null;

    public function __construct() {
        self::$openningState = new OpenningState();
        self::$closeingState = new ClosingState();
        self::$runningState =  new RunningState();
        self::$stoppingState = new StoppingState();

    }

    //定一个当前电梯状态
```

```
    private  $_liftState;

    public function getLiftState() {
        return $this->_liftState;
    }

    public function setLiftState($liftState) {
        $this->_liftState = $liftState;
        //把当前的环境通知到各个实现类中
        $this->_liftState->setContext($this);
    }


    public function open(){
        $this->_liftState->open();
    }

    public function close(){
        $this->_liftState->close();
    }

    public function run(){
        $this->_liftState->run();
    }

    public function stop(){
        $this->_liftState->stop();
    }
}

/**
 * 在电梯门开启的状态下能做什么事情
 */
class OpenningState extends LiftState {

    /**
     * 开启当然可以关闭了，我就想测试一下电梯门开关功能
     *
     */
    public function close() {
        //状态修改
        $this->_context->setLiftState(Context::$closeingState);
        //动作委托为CloseState来执行
        $this->_context->getLiftState()->close();
    }

    //打开电梯门
    public function open() {
        echo 'lift open...', '<br/>';
    }
    //门开着电梯就想跑，这电梯，吓死你！
    public function run() {
        //do nothing;
    }

    //开门还不停止？
    public function stop() {
        //do nothing;
    }

}

/**
 * 电梯门关闭以后，电梯可以做哪些事情
 */
class ClosingState extends LiftState {

    //电梯门关闭，这是关闭状态要实现的动作
    public function close() {
        echo 'lift close...', '<br/>';

    }
    //电梯门关了再打开，逗你玩呢，那这个允许呀
    public function open() {
        $this->_context->setLiftState(Context::$openningState);  //置为门敞状态
        $this->_context->getLiftState()->open();
    }

    //电梯门关了就跑，这是再正常不过了
    public function run() {
        $this->_context->setLiftState(Context::$runningState); //设置为运行状态;
        $this->_context->getLiftState()->run();
    }

    //电梯门关着，我就不按楼层

    public function stop() {
        $this->_context->setLiftState(Context::$stoppingState);  //设置为停止状态;
        $this->_context->getLiftState()->stop();
    }

}

/**
 * 电梯在运行状态下能做哪些动作
 */
class RunningState extends LiftState {

    //电梯门关闭？这是肯定了
    public function close() {
        //do nothing
    }

    //运行的时候开电梯门？你疯了！电梯不会给你开的
    public function open() {
        //do nothing
    }
```

```
    //这是在运行状态下要实现的方法
    public function run() {
        echo 'lift run...', '<br/>';
    }

    //这个事绝对是合理的，光运行不停止还有谁敢做这个电梯？！估计只有上帝了
    public function stop() {
        $this->_context->setLiftState(Context::$stoppingState); //环境设置为停止状态；
        $this->_context->getLiftState()->stop();
    }

}



/**
 * 在停止状态下能做什么事情
 */
class StoppingState extends LiftState {

    //停止状态关门？电梯门本来就是关着的！
    public function close() {
        //do nothing;
    }

    //停止状态，开门，那是要的！
    public function open() {
        $this->_context->setLiftState(Context::$openningState);
        $this->_context->getLiftState()->open();
    }
    //停止状态再跑起来，正常的很
    public function run() {
        $this->_context->setLiftState(Context::$runningState);
        $this->_context->getLiftState()->run();
    }
    //停止状态是怎么发生的呢？当然是停止方法执行了
    public function stop() {
        echo 'lift stop...', '<br/>';
    }

}

/**
 * 模拟电梯的动作
 */
class Client {

    public static function main() {
        $context = new Context();
        $context->setLiftState(new ClosingState());

        $context->open();
        $context->close();
        $context->run();
        $context->stop();
    }
}
Client::main();
```

9.与其他相关模式

1）职责链模式，

职责链模式和状态模式都可以解决If分支语句过多，

从定义来看，状态模式是一个对象的内在状态发生改变（一个对象，相对比较稳定，处理完一个对象下一个对象的处理一般都已确定），

而职责链模式是多个对象之间的改变（多个对象之间的话，就会出现某个对象不存在的现在，就像我们举例的公司请假流程，经理可能不在公司情况），这也说明他们两个模式处理的情况不同。

这两个设计模式最大的区别就是状态模式是让各个状态对象自己知道其下一个处理的对象是谁。

而职责链模式中的各个对象并不指定其下一个处理的对象到底是谁，只有在客户端才设定。

用我们通俗的编程语言来说，就是

状态模式：

相当于If else if else；

设计路线：各个State类的内部实现(相当于If，else If内的条件)

执行时通过State调用Context方法来执行。

职责链模式：

相当于Swich case

设计路线：客户设定，每个子类(case)的参数是下一个子类(case)。

使用时，向链的第一个子类的执行方法传递参数就可以。

就像对设计模式的总结，有的人采用的是状态模式，从头到尾，提前一定定义好下一个处理的对象是谁，而我采用的是职责链模式，随时都有可能调整链的顺序。

2） 策略模式：（http://www.cnblogs.com/Mainz/archive/2007/12/15/996081.html）（状态模式是策略模式的孪生兄弟）

状态模式和策略模式的实现方法非常类似，都是利用多态把一些操作分配到一组相关的简单的类中，因

此很多人认为这两种模式实际上是相同的。

然而在现实世界中，策略（如促销一种商品的策略）和状态（如同一个按钮来控制一个电梯的状态，又如手机界面中一个按钮来控制手机）是两种完全不同的思想。当我们对状态和策略进行建模时，这种差异会导致完全不同的问题。例如，对状态进行建模时，状态迁移是一个核心内容；然而，在选择策略时，迁移与此毫无关系。另外，策略模式允许一个客户选择或提供一种策略，而这种思想在状态模式中完全没有。

一个策略是一个计划或方案，通过执行这个计划或方案，我们可以在给定的输入条件下达到一个特定的目标。策略是一组方案，他们可以相互替换；选择一个策略，获得策略的输出。策略模式用于随不同外部环境采取不同行为的场合。我们可以参考微软企业库底层Object Builder的创建对象的strategy实现方式。而状态模式不同，对一个状态特别重要的对象，通过状态机来建模一个对象的状态；状态模式处理的核心问题是状态的迁移，因为在对象存在很多状态情况下，对各个business flow，各个状态之间跳转和迁移过程都是及其复杂的。

例如一个工作流，审批一个文件，存在新建、提交、已修改、HR部门审批中、老板审批中、HR审批失败、老板审批失败等状态，涉及多个角色交互，涉及很多事件，这种情况下用状态模式(状态机)来建模更加合适；把各个状态和相应的实现步骤封装成一组简单的继承自一个接口或抽象类的类，通过另外的一个Context来操作他们之间的自动状态变换，通过event来自动实现各个状态之间的跳转。在整个生命周期中存在一个状态的迁移曲线，这个迁移曲线对客户是透明的。我们可以参考微软最新的WWF 状态机工作流实现思想。

在状态模式中，状态的变迁是由对象的内部条件决定，外界只需关心其接口，不必关心其状态对象的创建和转化；

而策略模式里，采取何种策略由外部条件(C)决定。

他们应用场景（目的）却不一样，State模式重在强调对象内部状态的变化改变对象的行为，Strategy模式重在外部对策略的选择，策略的选择由外部条件决定，

也就是说算法的动态的切换。但由于它们的结构是如此的相似，我们可以认为“状态模式是完全封装且自修改的策略模式”。即状态模式是封装对象内部的状态的，而策略模式是封装算法族的

10.总结与分析

状态模式的主要优点在于封装了转换规则，并枚举可能的状态，它将所有与某个状态有关的行为放到一个类中，并且可以方便地增加新的状态，只需要改变对象状态即可改变对象的行为，还可以让多个环境对象共享一个状态对象，从而减少系统中对象的个数；其缺点在于使用状态模式会增加系统类和对象的个数，且状态模式的结构与实现都较为复杂，如果使用不当将导致程序结构和代码的混乱，对于可以切换状态的状态模式不满足“开闭原则”的要求。

版权声明：本文为博主原创文章，未经博主允许不得转载。

上一篇 设计模式 (十六) 观察者模式Observer (对象行为型)
下一篇 设计模式 (十八) 策略模式Strategy (对象行为型)

顶 17
踩 0

主题推荐 设计模式 对象 解决方案 应用程序 软件开发

猜你在找

iOS开发Swift语言学习教程
微信公众平台深度开发(Java版)
JDBC视频教程
Cocos2d-x实战-手把手教你上线项目-迷失航线
微信公众平台开发入门
Avatar状态机设计
手把手实现红黑树
去哪儿网面试问题
ehcache memcache redis 三大缓存男高音
Java内部类总结 吐血之作

准备好了么？跳吧 ！
更多职位尽在 CSDN JOB

视觉设计人员（美工） 我要跳槽
东软集团华东大区 | 9-18K/月

船舶设计软件开发工程技术人员 我要跳槽
上海交通大学船舶数字化工程中心 | 10-15K/月

产品经理（客户端原型及交互设计） 我要跳槽
国卫信安教育科技（北京）有限公司 | 10-15K/月

UI设计 我要跳槽
呼和浩特拓普管理咨询有限公司 | 15-30K/月

查看评论

13楼 kk80805588 2015-02-24 04:38发表

http://www.nowamagic.net/librarys/veda/detail/1605

请搂着看看这边关于状态模式的代码,好像和你这个效果不一样, 我看设计模式的树上介绍的和我贴的网址类似

12楼 十一期张振华 2015-02-23 15:53发表

您好，请问您的UML图是用什么工具画的？

11楼 老哂 2015-02-09 16:48发表

写的很好，感谢分享。

10楼 tsj11514oo 2014-10-01 15:24发表

学习啦，总结的很全，尤其区别那最好。

9楼 john-huang 2014-09-19 09:12发表

学习，很具体，例子也很不错

8楼 全未觉醒 2014-01-12 13:55发表

> 引用“woshichu66”的评论：
> 讲的很好，顶一个！

7楼 zhuojiajin 2013-12-05 08:55发表

茅塞顿开！！！

6楼 blueboy0722 2013-10-30 16:42发表

请教个问题啊
$context->setLiftState(new ClosingState());
为什么不写成
$context->setLiftState(Context::$stoppingState);

Re: 真实的归宿 2013-10-30 17:20发表

回复blueboy0722：$context->setLiftState(new ClosingState());
当然不能写成：
$context->setLiftState(Context::$stoppingState);

new ClosingState()是一个新的对象。 new ClosingState()和Context::$stoppingState是两个完全不同的对象。

Re: blueboy0722 2013-10-30 18:16发表

回复hguisu：但是感觉new ClosingState()这个对象以后就没有用处了，直接用Context::$stoppingState有何不可的呢

Re: 真实的归宿 2013-10-30 19:58发表

回复blueboy0722：恩，你说的也有理。
其实new ClosingState()这个对象相对于，我们要测试电梯。首先肯定是关电梯。

5楼 tachen 2013-09-06 19:53发表

学习了，那个我想问下，如果按照状态模式来做的话，每增加一种状态，每个state依旧要增加相应的代码是这样理解的不？？？？

Re: Astah 2014-11-20 12:09发表

回复tachen：我也感觉像是，比如增加一种状态，pause，那么就需要在closeState、runState、stopState、openState这几个类中分别都要增加一个方法，pause()。。。

4楼 ljhabc1982 2013-09-03 17:11发表

讲的很好，很清楚

3楼 田有朋 2013-07-24 19:33发表

很好，学习啦

2楼 woshichu66 2013-07-12 10:44发表

讲的很好，顶一个！

1楼 FingerStyle 2013-01-27 17:18发表

写的不错，用心了。

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

京 ICP 证 070598 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved