/\*

核心层： 控制类

\*/

using System;

using System.Collections.Generic;

using PureMVC.Interfaces;

using PureMVC.Patterns;

namespace PureMVC.Core{

public class **Controller** : IController{

protected IView **m\_view**; //IView 的引用

protected IDictionary<string, Type> **m\_commandMap**; //Command 类引用的（通知名）映射

protected static volatile IController m\_instance; //接口实例

protected readonly object m\_syncRoot = new object();//锁对象

protected static readonly object m\_staticSyncRoot = new object();//静态锁

protected Controller(){

m\_commandMap = new Dictionary<string, Type>();

InitializeController(); //初始化

}

static Controller() {}

public static IController Instance{

get{

if (m\_instance == null){

lock (m\_staticSyncRoot){

if (m\_instance == null) m\_instance = new Controller();

}

}

return m\_instance;

}

}

protected virtual void **InitializeController**(){

m\_view = View.Instance;

}

public virtual void **ExecuteCommand**(INotification note){

Type commandType = null;

lock (m\_syncRoot){

if (!m\_commandMap.ContainsKey(note.Name)) return;

commandType = m\_commandMap[note.Name];

}

object commandInstance = Activator.CreateInstance(commandType);

if (commandInstance is ICommand){

((ICommand) commandInstance).Execute(note);

}

}

public virtual void **RegisterCommand**(string notificationName, Type commandType){

lock (m\_syncRoot){

if (!m\_commandMap.ContainsKey(notificationName)){

m\_view.RegisterObserver(notificationName, new Observer("executeCommand", this));

}

m\_commandMap[notificationName] = commandType;

}

}//RegisterCommand\_end

public virtual bool HasCommand(string notificationName){

lock (m\_syncRoot){

return m\_commandMap.ContainsKey(notificationName);

}

}

public virtual void RemoveCommand(string notificationName){

lock (m\_syncRoot){

if (m\_commandMap.ContainsKey(notificationName)){

m\_view.RemoveObserver(notificationName, this);

m\_commandMap.Remove(notificationName);

}

}

}

}//Class\_end

}

/\*

核心层： 模型类

\*/

namespace PureMVC.Core{

public class Model : IModel{

protected static volatile IModel m\_instance; //本类实例

protected IDictionary<string, IProxy> **m\_proxyMap**; //代理集合类

protected readonly object m\_syncRoot = new object();//同步锁定对象（配合Lock关键字）

protected static readonly object m\_staticSyncRoot = new object();//静态同步锁定对象

protected Model(){

m\_proxyMap = new Dictionary<string, IProxy>();

InitializeModel();

}

static Model(){}

public static IModel Instance{

get{

if (m\_instance == null){

lock (m\_staticSyncRoot){

if (m\_instance == null) m\_instance = new Model();

}

}

return m\_instance;

}

}

protected virtual void InitializeModel(){}

public virtual void **RegisterProxy**(IProxy proxy){

lock (m\_syncRoot){

m\_proxyMap[proxy.ProxyName] = proxy;

}

proxy.OnRegister();

}

public virtual IProxy **RetrieveProxy**(string proxyName){

lock (m\_syncRoot){

if (!m\_proxyMap.ContainsKey(proxyName)) return null;

return m\_proxyMap[proxyName];

}

}

public virtual bool HasProxy(string proxyName){

lock (m\_syncRoot){

return m\_proxyMap.ContainsKey(proxyName);

}

}

public virtual IProxy **RemoveProxy**(string proxyName){

IProxy proxy = null;

lock (m\_syncRoot){

if (m\_proxyMap.ContainsKey(proxyName)){

proxy = RetrieveProxy(proxyName);

m\_proxyMap.Remove(proxyName);

}

}

if (proxy != null) proxy.OnRemove();

return proxy;

}

}

}

/\*

核心类： 视图层

\*/

namespace PureMVC.Core{

public class View : IView{

protected IDictionary<string, IMediator> **m\_mediatorMap**;//缓存IMediator实例集合

protected IDictionary<string, IList<IObserver>> ***m\_observerMap***;

protected static volatile IView m\_instance;

protected readonly object m\_syncRoot = new object();

protected static readonly object m\_staticSyncRoot = new object();

protected View(){

m\_mediatorMap = new Dictionary<string, IMediator>();

m\_observerMap = new Dictionary<string, IList<IObserver>>();

InitializeView();

}

static View(){}

public static IView Instance{

get

{

if (m\_instance == null)

{

lock (m\_staticSyncRoot)

{

if (m\_instance == null) m\_instance = new View();

}

}

return m\_instance;

}

}

protected virtual void InitializeView(){ }

public virtual void **RegisterObserver**(string notificationName, IObserver observer){

lock (m\_syncRoot){

if (!m\_observerMap.ContainsKey(notificationName)){

m\_observerMap[notificationName] = new List<IObserver>();

}

m\_observerMap[notificationName].Add(observer);

}

}

public virtual void **NotifyObservers**(INotification notification){

IList<IObserver> observers = null;

lock (m\_syncRoot){

if (m\_observerMap.ContainsKey(notification.Name)){

IList<IObserver> observers\_ref = m\_observerMap[notification.Name];

observers = new List<IObserver>(observers\_ref);

}

}

if (observers != null){

// Notify Observers from the working array

for (int i = 0; i < observers.Count; i++){

IObserver observer = observers[i];

observer.NotifyObserver(notification);

}

}

}

public virtual void **RemoveObserver**(string notificationName, object notifyContext){

lock (m\_syncRoot){

if (m\_observerMap.ContainsKey(notificationName)){

IList<IObserver> observers = m\_observerMap[notificationName];

for (int i = 0; i < observers.Count; i++){

if (observers[i].CompareNotifyContext(notifyContext)){

observers.RemoveAt(i);

break;

}

}

if (observers.Count == 0){

m\_observerMap.Remove(notificationName);

}

}

}

}

public virtual void **RegisterMediator**(IMediator mediator){

lock (m\_syncRoot){

if (m\_mediatorMap.ContainsKey(mediator.MediatorName)) return;

m\_mediatorMap[mediator.MediatorName] = mediator;

IList<string> interests = mediator.ListNotificationInterests();

if (interests.Count > 0){

IObserver observer = new Observer("handleNotification", mediator);

for (int i = 0; i < interests.Count; i++){

RegisterObserver(interests[i].ToString(), observer);

}

}

}

mediator.OnRegister();

}

public virtual IMediator **RetrieveMediator**(string mediatorName)

{

lock (m\_syncRoot)

{

if (!m\_mediatorMap.ContainsKey(mediatorName)) return null;

return m\_mediatorMap[mediatorName];

}

}

public virtual IMediator **RemoveMediator**(string mediatorName){

IMediator mediator = null;

lock (m\_syncRoot)

{

if (!m\_mediatorMap.ContainsKey(mediatorName)) return null;

mediator = (IMediator) m\_mediatorMap[mediatorName];

IList<string> interests = mediator.ListNotificationInterests();

for (int i = 0; i < interests.Count; i++){

RemoveObserver(interests[i], mediator);

}

m\_mediatorMap.Remove(mediatorName);

}

if (mediator != null) mediator.OnRemove();

return mediator;

}

public virtual bool HasMediator(string mediatorName)

{

lock (m\_syncRoot)

{

return m\_mediatorMap.ContainsKey(mediatorName);

}

}

}

}

/\*

Facade 核心入口类

\*/

namespace PureMVC.Patterns{

public class **Facade** : IFacade{

protected IController **m\_controller**;

protected IModel **m\_model**;

protected IView **m\_view**;

protected static volatile IFacade m\_instance;

protected static readonly object m\_staticSyncRoot = new object();

protected Facade() {

InitializeFacade();

}

public static IFacade Instance{

get{

if (m\_instance == null){

lock (m\_staticSyncRoot){

if (m\_instance == null) m\_instance = new Facade();

}

}

return m\_instance;

}

}

static Facade(){}

protected virtual void **InitializeFacade**(){

InitializeModel();

InitializeController();

InitializeView();

}

public virtual void **RegisterProxy**(IProxy proxy){

m\_model.RegisterProxy(proxy);

}

public virtual IProxy **RetrieveProxy**(string proxyName){

return m\_model.RetrieveProxy(proxyName);

}

public virtual IProxy **RemoveProxy**(string proxyName){

return m\_model.RemoveProxy(proxyName);

}

public virtual bool HasProxy(string proxyName){

return m\_model.HasProxy(proxyName);

}

public virtual void **RegisterCommand**(string notificationName, Type commandType){

m\_controller.RegisterCommand(notificationName, commandType);

}

public virtual void **RemoveCommand**(string notificationName){

m\_controller.RemoveCommand(notificationName);

}

public virtual bool HasCommand(string notificationName){

return m\_controller.HasCommand(notificationName);

}

public virtual void **RegisterMediator**(IMediator mediator){

m\_view.RegisterMediator(mediator);

}

public virtual IMediator **RetrieveMediator**(string mediatorName){

return m\_view.RetrieveMediator(mediatorName);

}

public virtual IMediator **RemoveMediator**(string mediatorName){

return m\_view.RemoveMediator(mediatorName);

}

public virtual bool HasMediator(string mediatorName){

return m\_view.HasMediator(mediatorName);

}

public virtual void **NotifyObservers**(INotification notification){

m\_view.NotifyObservers(notification);

}

public virtual void **SendNotification**(string notificationName){

NotifyObservers(new Notification(notificationName));

}

public virtual void **SendNotification**(string notificationName, object body){

NotifyObservers(new Notification(notificationName, body));

}

public virtual void **SendNotification**(string notificationName, object body, string type){

NotifyObservers(new Notification(notificationName, body, type));

}

protected virtual void **InitializeController**(){

if (m\_controller != null) return;

m\_controller = Controller.Instance;

}

protected virtual void **InitializeModel**(){

if (m\_model != null) return;

m\_model = Model.Instance;

}

protected virtual void **InitializeView**(){

if (m\_view != null) return;

m\_view = View.Instance;

}

}

}

/\*\*\*

\* 简单命令类

\*/

namespace PureMVC.Patterns{

public class SimpleCommand : Notifier, ICommand, INotifier{

public virtual void **Execute**(INotification notification){}

}

}

/\*

\* 大量命令类

\*/

namespace PureMVC.Patterns{

public class MacroCommand : Notifier, ICommand, INotifier{

private IList<Type> **m\_subCommands**;

public MacroCommand(){

m\_subCommands = new List<Type>();

InitializeMacroCommand();

}

public virtual void **Execute**(INotification notification){

while (m\_subCommands.Count > 0){

Type commandType = m\_subCommands[0];

object commandInstance = Activator.CreateInstance(commandType);

if (commandInstance is ICommand){

((ICommand) commandInstance).Execute(notification);

}

m\_subCommands.RemoveAt(0);

}

}

protected virtual void InitializeMacroCommand(){}

protected void AddSubCommand(Type commandType){

m\_subCommands.Add(commandType);

}

}

}

/\*\*\*

\* 视图类

\*/

namespace PureMVC.Patterns{

public class **Mediator** : Notifier, IMediator, INotifier{

protected string m\_mediatorName;

protected object m\_viewComponent;

public const string NAME = "Mediator";

public virtual string MediatorName{

get { return m\_mediatorName; }

}

public virtual object ViewComponent{

get { return m\_viewComponent; }

set { m\_viewComponent = value; }

}

public Mediator()

: this(NAME, null){}

public Mediator(string mediatorName)

: this(mediatorName, null){}

public Mediator(string mediatorName, object viewComponent){

m\_mediatorName = (mediatorName != null) ? mediatorName : NAME;

m\_viewComponent = viewComponent;

}

public virtual IList<string> **ListNotificationInterests**(){

return new List<string>();

}

public virtual void **HandleNotification**(INotification notification){}

public virtual void **OnRegister**(){}

public virtual void **OnRemove**(){}

}

}

/\*

通知类

\*/

namespace PureMVC.Patterns{

public class **Notification** : INotification{

private string m\_name;

private string m\_type;

private object m\_body;

public virtual string Name{

get { return m\_name; }

}

public virtual object Body{

get{

return m\_body;

}

set{

m\_body = value;

}

}

public virtual string Type{

get{

return m\_type;

}

set{

m\_type = value;

}

}

public Notification(string name)

: this(name, null, null){ }

public Notification(string name, object body)

: this(name, body, null){ }

public Notification(string name, object body, string type){

m\_name = name;

m\_body = body;

m\_type = type;

}

public override string ToString(){

string msg = "Notification Name: " + Name;

msg += "\nBody:" + ((Body == null) ? "null" : Body.ToString());

msg += "\nType:" + ((Type == null) ? "null" : Type);

return msg;

}

}

}

/\*

通知者类

\*/

using System;

using PureMVC.Interfaces;

using PureMVC.Patterns;

namespace PureMVC.Patterns

{

public class **Notifier** : INotifier

{

private IFacade **m\_facade** = PureMVC.Patterns.Facade.Instance;

protected IFacade Facade{

get { return m\_facade; }

}

public virtual void SendNotification(string notificationName) {

m\_facade.SendNotification(notificationName);

}

public virtual void SendNotification(string notificationName, object body){

m\_facade.SendNotification(notificationName, body);

}

public virtual void SendNotification(string notName, object body, string type){

m\_facade.SendNotification(notName, body, type);

}

}

}

/\* 观察者类 \*/

namespace PureMVC.Patterns{

public class **Observer** : IObserver{

private string **m\_notifyMethod**; //通知方法名称

private object **m\_notifyContext**; //通知上下文

protected readonly object m\_syncRoot = new object(); //锁定

public virtual string NotifyMethod{

private get{

return m\_notifyMethod;

}

set{

m\_notifyMethod = value;

}

}

public virtual object NotifyContext{

private get{

return m\_notifyContext;

}

set{

m\_notifyContext = value;

}

}

public **Observer**(string notifyMethod, object notifyContext){

m\_notifyMethod = notifyMethod;

m\_notifyContext = notifyContext;

}

public virtual void **NotifyObserver**(INotification notification){

object context;

string method;

lock (m\_syncRoot){

context = NotifyContext;

method = NotifyMethod;

}

Type t = context.GetType();

BindingFlags f = BindingFlags.Instance | BindingFlags.Public | BindingFlags.IgnoreCase;

MethodInfo mi = t.GetMethod(method, f);

mi.Invoke(context, new object[] { notification });

}

public virtual bool CompareNotifyContext(object obj){

lock (m\_syncRoot){

return NotifyContext.Equals(obj);

}

} }}

/\*

代理类

\*/

using System;

using PureMVC.Interfaces;

using PureMVC.Patterns;

namespace PureMVC.Patterns{

public class **Proxy** : Notifier, IProxy, INotifier{

protected string m\_proxyName;

protected object m\_data;

public static string NAME = "Proxy";

public virtual string ProxyName{

get { return m\_proxyName; }

}

public virtual object Data{

get { return m\_data; }

set { m\_data = value; }

}

public Proxy()

: this(NAME, null){}

public Proxy(string proxyName)

: this(proxyName, null){}

public Proxy(string proxyName, object data){

m\_proxyName = (proxyName != null) ? proxyName : NAME;

if (data != null) m\_data = data;

}

public virtual void OnRegister(){}

public virtual void OnRemove(){}

}

}

------------------------------------------(接口)--------------------------------------------

namespace PureMVC.Interfaces

{

public interface **ICommand**

{

void Execute(INotification notification);

}

}

namespace PureMVC.Interfaces{

public interface **IProxy**

{

string ProxyName { get; }

object Data { get; set; }

void OnRegister();

void OnRemove();

}

}

namespace PureMVC.Interfaces{

public interface **IView**{

void RegisterObserver(string notificationName, IObserver observer);

void RemoveObserver(string notificationName, object notifyContext);

void NotifyObservers(INotification note);

void RegisterMediator(IMediator mediator);

IMediator RetrieveMediator(string mediatorName);

IMediator RemoveMediator(string mediatorName);

bool HasMediator(string mediatorName);

}

}

namespace PureMVC.Interfaces{

public interface **IObserver**{

string NotifyMethod { set; }

object NotifyContext { set; }

void NotifyObserver(INotification notification);

bool CompareNotifyContext(object obj);

}

}

namespace PureMVC.Interfaces{

public interface **INotifier**{

void SendNotification(string notificationName);

void SendNotification(string notificationName, object body);

void SendNotification(string notificationName, object body, string type);

}

}

namespace PureMVC.Interfaces{

public interface **INotification**{

string Name { get; }

object Body { get; set; }

string Type { get; set; }

string ToString();

}

}

namespace PureMVC.Interfaces{

public interface **IModel**{

void RegisterProxy(IProxy proxy);

IProxy RetrieveProxy(string proxyName);

IProxy RemoveProxy(string proxyName);

bool HasProxy(string proxyName);

}

}

namespace PureMVC.Interfaces{

public interface **IMediator**{

string MediatorName { get; }

object ViewComponent { get; set; }

IList<string> ListNotificationInterests();

void HandleNotification(INotification notification);

void OnRegister();

void OnRemove();

}

}

namespace PureMVC.Interfaces{

public interface **IFacade** : INotifier{

void RegisterProxy(IProxy proxy);

IProxy RetrieveProxy(string proxyName);

IProxy RemoveProxy(string proxyName);

bool HasProxy(string proxyName);

void RegisterCommand(string notificationName, Type commandType);

void RemoveCommand(string notificationName);

bool HasCommand(string notificationName);

void RegisterMediator(IMediator mediator);

IMediator RetrieveMediator(string mediatorName);

IMediator RemoveMediator(string mediatorName);

bool HasMediator(string mediatorName);

void NotifyObservers(INotification note);

}

}

namespace PureMVC.Interfaces{

public interface **IController**{

void RegisterCommand(string notificationName, Type commandType);

void ExecuteCommand(INotification notification);

void RemoveCommand(string notificationName);

bool HasCommand(string notificationName);

}

}