The Client of FinanX Stocker System

The Document of Architecture Description

**Document update history**

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人员 | 日期 | 变更原因 | 版本号 |
| 陈乾明 | 2014/08/12 | creation&&draft | 1.0 |
| 陈乾明 | 2014/08/16 | Add module interface | 1.1 |
| 陈乾明 | 2014/08/28 | Interface correction | 1.2 |

Catalog

[1. Introduction 4](#_Toc398411245)

[1.1 Compilation Purpose 4](#_Toc398411246)

[2. Architecture Model 4](#_Toc398411247)

[2.1 The Perspective of Logic 4](#_Toc398411248)

[2.2 Application Process Step 6](#_Toc398411249)

[2.3 Package Design 6](#_Toc398411250)

[2.4 Deployment 9](#_Toc398411251)

[3. The Perspective of Interface 9](#_Toc398411252)

[3.1 Host Application Module 9](#_Toc398411253)

[3.2 Infrastructure Module 11](#_Toc398411254)

[3.3 Charts Module 12](#_Toc398411255)

[3.4 Login Module 13](#_Toc398411256)

[3.5 Market Module 14](#_Toc398411257)

[3.6 Navigator Module 15](#_Toc398411258)

[3.7 Statistics Module 15](#_Toc398411259)

[3.8 Trade Module 16](#_Toc398411260)

[3.9 Watch Module 16](#_Toc398411261)

[3.10 Wizard Module 17](#_Toc398411262)

[4. The Perspective of Information 18](#_Toc398411263)

[5. UI Design 20](#_Toc398411264)

[6. System Architecture Design Thinking 20](#_Toc398411265)

[6.1 C/S Responsibility Assignment 20](#_Toc398411266)

[6.2 WPF UI Framework 21](#_Toc398411267)

[6.3 Prism A Set of Assets for Building Complex WPF Applications 21](#_Toc398411268)

[6.4 MEF Managed Extensibility Framework 21](#_Toc398411269)

[6.5 MVVM Model 21](#_Toc398411270)

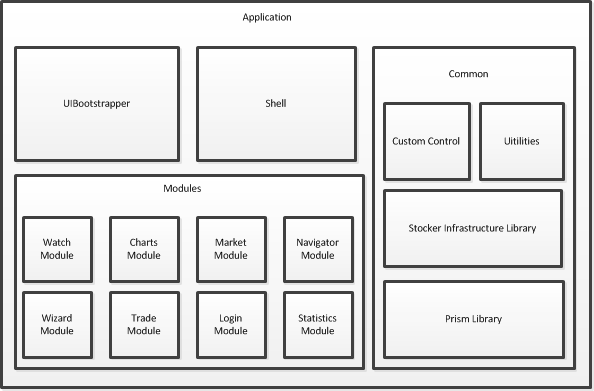
1. Introduction
   1. Compilation Purpose

This document describes the software design of the FinanX program of financial trading system, to guide the detailed design and development purposes, while realizing the communication with testing personnel.

This document is written for developers, testers and end users, is the navigation of understanding system.

1. Architecture Model
   1. The Perspective of Logic

Stocker uses WPF UI framework, Prism development framework, and MVVM mode.

The following illustration shows a high-level logical architecture view of the Stock Trader RI. 

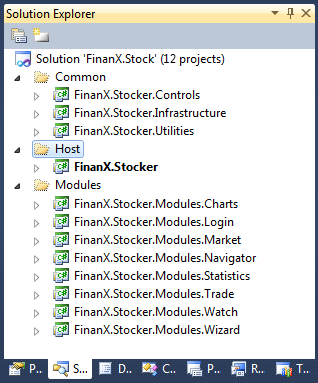
Picture 1 Finan-X Stocker Logic Achitecture

The following describes the main elements of the Stock Trader RI architecture:

Application：The application is lightweight and contains the shell that hosts each of the different UI components within the reference implementation. It also contains the StockTraderRIBootstrapper, which sets up the container and initializes module loading.

Modules: The solution is divided into the following eight modules, which are each maintained by separate teams in different locations:

* Watch Module: The Watch module contains the **Watch List** and **Add To Watch Stock List** functionality.
* Charts Module：The module includes Candle Stick, Volume Line, OHLC, Time Line reports, to view the stock market.
* Market Module：The module includes the stock market situation of Market Summary list, Stock Summary specific stock information.
* Navigator Module：The module includes a home page Index and a navigation bar of program function .
* Wizard Module：This module contains the application settings interface
* Trade Module：The module is mainly responsible for the acquisition of stock data, including stock list of real-time data, historical stock data, data classification.
* Login Module：The login module, including the Login View and login logic.
* Statistics Module: Statistical analysis module, the module includes the combination of interface in which the calculation of stock purchase is displayed, and analysis of stock returns logic.



* 1. Application Process Step



图 2Application Process Step

Application Process Step

1. The application uses the StockTraderRIBootstrapper, which inherits from the Prism Library's MefBootstrappe**r** for its initialization.
2. The application initializes the Prism Library's MefServiceLocatorAdapter for use in the modules.
3. The StockTraderRIBootstrapper creates and shows the shell view.
4. The Prism Library's ModuleCatalog finds all the modules the application needs to load.
5. The Prism Library's ModuleManager loads and initializes each module.
6. Modules use the Prism Library's RegionManager service to add a view to a region.
7. The Prism Library's Region displays the view.
   1. Package Design

Public Library dependencies：

Table 1 Public Library dependencies

|  |
| --- |
| Public Library dependencies |
| Microsoft.CSharp,  Presentation.Core,  PresentationFramework,  System,  System.Core,  System.Data,  System.Data.DataSetEctensions,  System.Xml,  System.Xml.Linq |

Table 2 Package Design

|  |  |  |
| --- | --- | --- |
|  | Namespace definition | Other library dependencies |
| Common | **FinanX.Stocker.Controls** | System.Xaml, WindowBase |
| **FinanX.Stocker.Utilities** | MySql.Data,  Newtonsoft.Json,  System.ComponentModel.Composition, System.Xaml, WindowBase |
| **FinanX.Stocker.Infrastructure** | Microsoft.Practices.Prism,  System.ComponentModel.Composition |
| Host | **FinanX.Stocker.Host** | System.Xaml,  System.Drawing,  WindowBase,  System.ComponentModel.Composition,  Microsoft.Practices.Prism,  Microsoft.Practices.Prism.MefExtensions,  FinanX.Stocker.Controls  FinanX.Stocker.Utilities  FinanX.Stocker.Infrastructure  FinanX.Stocker.Modules.Charts  FinanX.Stocker.Modules.Login  FinanX.Stocker.Modules.Market  FinanX.Stocker.Modules.Navigator  FinanX.Stocker.Modules.Statistics  FinanX.Stocker.Modules.Trade  FinanX.Stocker.Modules.Watch |
| Modules | **FinanX.Stocker.Modules.Charts** | Microsoft.Practices.Prism,  System.ComponentModel.Composition,  AmCharts.Windows.Stock,FinanX.Stocker.Infrastructure,  System.Xaml, WindowBase |
| **FinanX.Stocker.Modules.Login** | System.Xaml,  WindowBase,  System.ComponentModel.Composition,  Newtonsoft.Json,  FinanX.Stocker.Controls,  FinanX.Stocker.Utilities  FinanX.Stocker.Infrastructure |
| **FinanX.Stocker.Modules.Market** | System.Xaml,  WindowBase,  System.ComponentModel.Composition,  Microsoft.Practices.Prism,  Microsoft.Practices.Prism.MefExtensions,  FinanX.Stocker.Utilities  FinanX.Stocker.Infrastructure |
| **FinanX.Stocker.Modules.Navigator** | System.Xaml,  WindowBase,  System.ComponentModel.Composition,  Microsoft.Practices.Prism,  FinanX.Stocker.Utilities  FinanX.Stocker.Infrastructure |
| **FinanX.Stocker.Modules.Statistics** | System.Xaml,  WindowBase,  System.ComponentModel.Composition,  Microsoft.Practices.Prism,  FinanX.Stocker.Utilities  FinanX.Stocker.Infrastructure |
| **FinanX.Stocker.Modules.Trade** | System.Xaml,  WindowBase,  System.ComponentModel.Composition,  Newtonsoft.Json,  Microsoft.Practices.Prism,  FinanX.Stocker.Utilities  FinanX.Stocker.Infrastructure |
| **FinanX.Stocker.Modules.Watch** | System.Xaml,  WindowBase,  System.ComponentModel.Composition,  Microsoft.Practices.Prism,  FinanX.Stocker.Utilities  FinanX.Stocker.Infrastructure |
| **FinanX.Stocker.Modules.Wizard** | System.Xaml,  WindowBase,  System.ComponentModel.Composition,  Microsoft.Practices.Prism,  FinanX.Stocker.Utilities  FinanX.Stocker.Infrastructure |

* 1. Deployment



图 3Deployment

1. The Perspective of Interface
   1. Host Application Module

Responsibility of Host Module

|  |  |
| --- | --- |
| component | responsibility |
| App | The main program entrance |
| UIBootstrapper | UI starter, used to set the MEF container and initializing and loading function module |
| Shell | The main window Shell，mark implanted region |
| ShellViewModel | The data model of Shell View, is used to store the data displayed in the interface |
| CallbackLogger | Procedure logger |
| Splasher | Splashed window controller |
| Splash View | Splashed window |
| DispatcherHelper | The tool for the transmission of message in the Splashed window |
| MessageListener | Message listener of Splashed Window |
| LoginShell | The shell of Login Window |

Interface specification of Host Module as shown in Table 3:

Table 3 Interface specification of UI

|  |  |  |
| --- | --- | --- |
|  | | |
| App. OnStartup | syntax | protected override void OnStartup(StartupEventArgs e) |
| precondition | Program Start |
| postconditions | Start Splash , initialize the settings，  Control the possible exception triggered by AppDomain |
| UIBootstrapper. CreateShell | syntax | protected override DependencyObject CreateShell() |
| precondition | Shell View has joined MEF container |
| postconditions | Export Shell View |
| UIBootstrapper. CreateLogger | syntax | protected override ILoggerFacade CreateLogger() |
| precondition | Logger Examples have been added to the container |
| postconditions | Export Logger |
| UIBootstrapper. ConfigureAggregateCatalog | syntax | protected override void ConfigureAggregateCatalog() |
| precondition | Have add reference Module |
| postconditions | assemble Module set |
| UIBootstrapper.ConfigureDefaultRegionBehaviors | syntax | protected override IRegionBehaviorFactory ConfigureDefaultRegionBehaviors() |
| precondition | need to inject Attributes which has been used by View. |
| postconditions | Create automatic injection factory |
| Splasher.ShowSplash | syntax | public static void ShowSplash () |
| precondition | Program Start |
| postconditions | Show Splashed window |
| Splasher. CloseSplash | syntax | public static void CloseSplash () |
| precondition | Program initialised |
| postconditions | Close the splashed window |
| MessageListener.ReceiveMessage | syntax | public void ReceiveMessage ( string message ) |
| precondition | Message Property has been registered |
| postconditions | The Message property is updated, and displayed on the Splash |
| DispatcherHelper. DoEvents | syntax | public static void DoEvents ( ) |
| precondition | Program Start |
| postconditions | Using UI thread to transmission message |

* 1. Infrastructure Module

Responsibility of Infrastructure Module

|  |  |
| --- | --- |
| component | responsibility |
| AutoPopulateExportedViewsBehavior | 控制自动注入界面区域 |
| NaviItemSelectedEvent | 导航选择事件 |
| TickerSymbolSelectedEvent | 股票代号选择事件 |
| TradePricesUpdatedEvent | 股票数据更新事件 |
| ControlWindowEvent | 用户控件操作事件 |
| IMarketFeedService | 股票市场信息服务，用于提供股票概况 |
| IMarketHistoryService | 股票历史数据信息服务，用于提供股票历史数据 |
| ITradePositionService | 股票市场信息服务，用于提供股票实时详情 |
| IHeaderInfoProvider | 窗口标题信息interface |

interface

|  |  |  |
| --- | --- | --- |
|  | | |
| AutoPopulateExportedViewsBehavior. RegisteredViews | syntax | public Lazy<object, IViewRegionRegistration>[] RegisteredViews { get; set; } |
| precondition | View已使用ViewExport标识 |
| postconditions | 获取注册的视图 |
| IMarketFeedService.StockName | syntax | string GetStockName(string tickerSymbol); |
| precondition | 股票代号存在 |
| postconditions | 获得股票名称 |
| IMarketFeedService. GetPrice | syntax | decimal GetPrice(string tickerSymbol) |
| precondition | 股票代号存在 |
| postconditions | 获得股票价格 |
| IMarketFeedService. GetVolume | synax | long GetVolume(string tickerSymbol); |
| precondition | 股票代号存在 |
| postconditions | 获得股票成交量 |
| IMarketFeedService. SymbolExists | synax | bool SymbolExists(string tickerSymbol); |
| precondition | - |
| postconditions | 判断股票代号是否存在 |
| IMarketHistoryService. StockHistoryInfo | synax | ObservableCollection<TradeHistoryModel> GetStockHistoryInfo(string tikerSymbol) |
| precondition | 股票代号存在 |
| postconditions | 获得股票历史数据 |
| ITradePositionService. TradePosition | synax | AccountPosition GetTradePosition(string tikerSymbol); |
| precondition | 股票代号存在 |
| postconditions | 获得股票实时数据 |
| ITradePositionService. TradePositions | synax | List<AccountPosition> GetTradePositions(); |
| precondition | - |
| postconditions | 获得股票实时数据列表 |
| ITradePositionService. Updated | synax | event EventHandler<AccountPositionModelEventArgs> Updated; |
| precondition | 获取新股票数据 |
| postconditions | 股票数据被更新 |
| IHeaderInfoProvider.HeadInfo | synax | T HeaderInfo |
| precondition | - |
| postconditions | 获取视图标题 |

* 1. Charts Module

Charts Module的responsibility

|  |  |
| --- | --- |
| component | responsibility |
| CandleStickView | 蜡烛图（K线）视图 |
| CandleStickViewModel | 蜡烛图（K线）视图模型 |

interface

|  |  |  |
| --- | --- | --- |
|  | | |
| CandleStickViewModel. HeaderInfo | synax | public string HeaderInfo { get; set; } |
| precondition | 捕获到TickerSymbolSelectedEvent |
| postconditions | 提供视图标题信息 |
| CandleStickViewModel. Data | synax | public ObservableCollection<TradeHistoryModel> Data { get; set; } |
| precondition | 捕获到TickerSymbolSelectedEvent |
| postconditions | 为视图提供股票历史数据信息 |
| CandleStickViewModel. StockName | synax | public String StockName { get; set; } |
| precondition | 捕获到TickerSymbolSelectedEvent |
| postconditions | 为视图提供股票标题信息 |

* 1. Login Module

Login Module的responsibility

|  |  |
| --- | --- |
| component | responsibility |
| AccountView | 用于用户登录信息控件 |
| AccountViewModel | 用于用户登录信息控件数据模型 |
| ILoginController | 登录控制器 |
| LoginView | 登录界面控件 |
| LoginViewModel | 登录界面数据模型 |
| WizardView | 登录配置窗口 |
| WizardViewModel | 登录配置窗口数据模型 |
| LoginResult | 登录返回消息 |
| UserInfoModel | 用户信息模型 |

interface

|  |  |  |
| --- | --- | --- |
|  | | |
| ILoginController.Login | synax | LoginResult Login(string userEmail, string password, bool? isAutoLogin); |
| precondition | 账户，密码不为空，网络正常 |
| postconditions | 用户登录成功或失败 |
| ILoginController. AutoLogin | synax | LoginResult AutoLogin(string userEmail, string autoLoginKey); |
| precondition | Autologinkey和用户名不为空 |
| postconditions | 用户登录成功或失败 |
| ILoginController.Logout | synax | DefaultTransportResult Logout(string userEmail); |
| precondition | 用户名不为空 |
| postconditions | 用户注销登录成功或失败 |
| LoginViewModel. ControlWindowCommand | synax | public ICommand ControlWindowCommand  {get ; private set ; } |
| precondition | 触发界面控制器命令 |
| postconditions | 执行控制命令 |
| LoginViewModel.LoginCommand | synax | public ICommand LoginCommand {get; private set} |
| precondition | 触发登录命令 |
| postconditions | 执行登录动作，调用LoginController.login |

* 1. Market Module

Market Module的responsibility

|  |  |
| --- | --- |
| component | responsibility |
| IMarketSummaryViewModel | 股票市场概况列表控件模型 |
| IObservableMarket | 股票市场列表数据 |
| MarketSummaryItem | 股票市场列表项 |
| MarketSummaryView | 股票市场概况列表控件 |
| StockSummaryView | 个股概况信息控件 |
| StockSummaryViewModel | 个股概况信息控件模型 |

interface

|  |  |  |
| --- | --- | --- |
|  | | |
| IObservableMarket. Items | synax | Public ObservableCollection<MarketSummaryItem> Items { get; } |
| precondition | -- |
| postconditions | 获取实时股票信息列表 |
| StockSummaryViewModel. CurrentTradePosition | synax | public AccountPosition CurrentTradePosition{get;private set;} |
| precondition | 选中股票实时列表中股票项 |
| postconditions | 获得当前选中股票的实时信息 |

* 1. Navigator Module

Navigator Module的responsibility

|  |  |
| --- | --- |
| component | responsibility |
| INavigateController | 导航控制器，用于发布导航选择事件 |
| StockListNavigatorView | 股票列表导航视图控件 |
| StockListNavigatorViewModel | 股票列表导航视图控件数据模型 |
| Index | 导航主页控件 |
| IndexViewModel | 导航主页控件数据模型 |

interface

|  |  |  |
| --- | --- | --- |
|  | | |
| INavigateController. NavigateCommand | synax | DelegateCommand<string> NavigateCommand { get; } |
| precondition | 触发导航事件 |
| postconditions | 跳转到相应的界面 |
| StockListNavigatorViewModel. CurrentStockItem | synax | public StockInfo CurrentStockItem |
| precondition | 导航栏中选中某个股票项 |
| postconditions | 获取当前选中股票项 |
| StockListNavigatorViewModel. Categories | synax | public IDictionary<string,string> GetCategories() |
| precondition | -- |
| postconditions | 获取所有股票分类目录 |
| StockListNavigatorViewModel. Stocks | synax | public ICollection<StockInfo> GetStocks() |
| precondition | -- |
| postconditions | 获取所有股票列表 |

* 1. Statistics Module

Statistics Module的responsibility

|  |  |
| --- | --- |
| component | responsibility |
| CompositionView | 股票组合计算控件 |
| CompositionViewModel | 股票组合计算控件模型 |

* 1. Trade Module

Trade Module的responsibility

|  |  |
| --- | --- |
| component | responsibility |
| TradeFeedService.cs | 股票市场信息服务，用于提供股票概况，提供HttpWeb Request实现 |
| TradeHositoryService.cs | 股票历史数据信息服务，用于提供股票历史数据，提供HttpWeb Request实现 |
| TradePositionService.cs | 股票市场信息服务，用于提供股票实时详情，提供HttpWeb Request实现 |

* 1. Watch Module

Watch Module的responsibility

|  |  |
| --- | --- |
| component | responsibility |
| WatchListView | 关注个股列表控件 |
| WatchListViewModel | 关注个股列表控件模型 |
| IWatchListService | 关注个股列表服务 |
| WatchItem | 关注个股列表项 |

interface

|  |  |  |
| --- | --- | --- |
|  | | |
| IWatchListService. RetrieveWatchList | synax | ObservableCollection<string> RetrieveWatchList(); |
| precondition | 查询关注列表 |
| postconditions | 获取关注股票列表 |
| IWatchListService. AddWatchCommand | synax | ICommand AddWatchCommand { get; set; } |
| precondition | -- |
| postconditions | 增加关注股票 |
| WatchListViewModel. WatchListItems | synax | public ObservableCollection<WatchItem> WatchListItems |
| precondition | -- |
| postconditions | 获取关注股票列表 |
| WatchListViewModel. CurrentWatchItem | synax | public WatchItem CurrentWatchItem |
| precondition | -- |
| postconditions | 获得当前关注项 |

* 1. Wizard Module

Wizard Module的responsibility

|  |  |
| --- | --- |
| component | responsibility |
| WizardView | 设置窗口控件 |
| WizardViewModel | 设置窗口控件模型 |

1. 信息视角

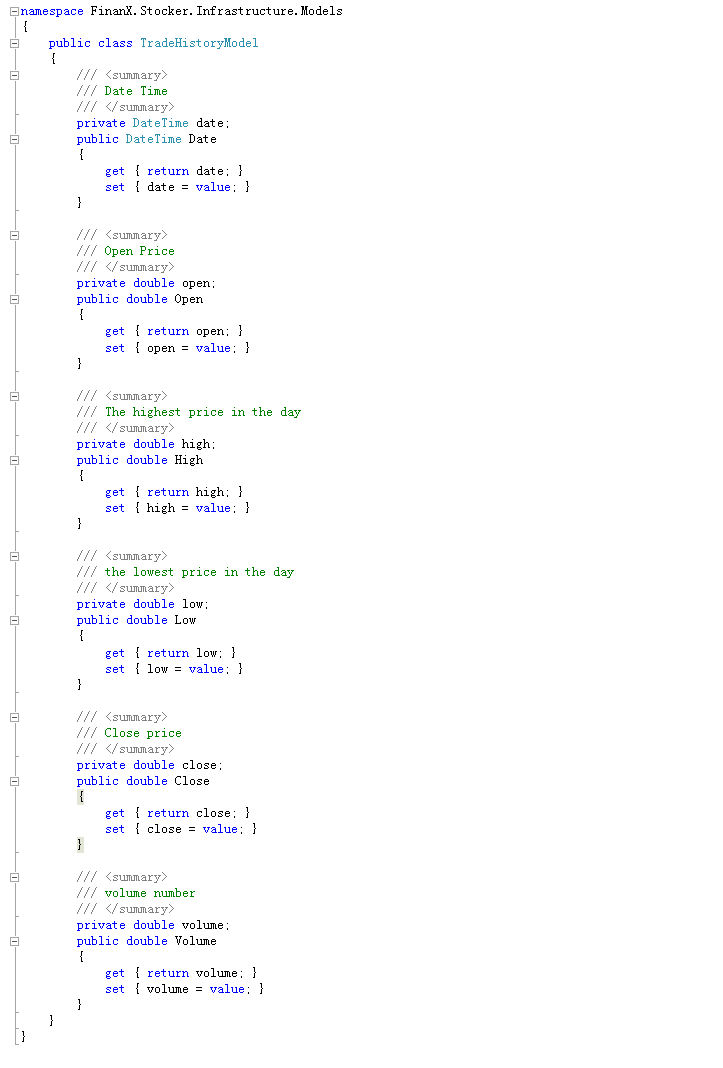


图 4Data Definition Example

1. 用户界面设计



图 5Login Window Concept View



图 6 Shell Concept View

1. 系统体系结构设计思路
   1. C/Sresponsibility分配

Client端中实现界面、简单逻辑判断，并通过Http web request向服务器发送远程请求；

Server端负责系统业务逻辑实现与数据存储范围，属于胖服务器，可降低业务逻辑的维护代价，提升系统的安全性，为PC客户端和Web页面提供服务。

* 1. WPF 用户界面框架

Windows Presentation Foundation (WPF)是Microsoft新一代图形系统，运行在.NET Framework 3.0及以上版本，为用户界面、2D/3D 图形、文档和媒体提供了统一的描述和操作方法。基于DirectX 9/10技术的WPF其图形向量渲染引擎也大大改进了传统的2D界面，它提供了超丰富的.NET UI 框架，集成了矢量图形，丰富的流动文字支持(flow text support)，3D视觉效果和控件模型框架。

* 1. Prism组合界面框架

Prism是组合式界面框架，它把界面（Interface）与逻辑相剥离，从而美工和Program员可以各按其事，互不干扰。把界面拆分成若干小的 Module（Module），从而使 Module间的交互实现了最大的松散耦合，降低了“牵一发而动全身”的风险。此外，在部署的时候，也可以实现按需（OnCommand）加载和更新。 Module的拆分，使得开发和测试也可以独立地进行。这是因为Prism使用了TDD的设计理念，当然我们在开发流程中也要follow这样的思想去编程，即“开发未动，测试先行”。

最大程度的实现了可复用（re-use）。尤其是日志关联和权限管理，是完全独立于主逻辑的（在数学上，这叫垂直关系，二者的叉积为零，从而各自改变而互不影响）。

* 1. MEF托管可扩展性框架

Managed Extensibility Framework（MEF）是.NET平台下的一个扩展性管理框架，它是一系列特性的集合，包括依赖注入(DI)以及Duck Typing等。MEF为开发人员提供了一个工具，让我们可以轻松的对应用Program进行扩展并且对已有的代码产生最小的影响，开发人员在开发过程中根据功能要求定义一些扩展点，之后就可以使用这些扩展点与应用Program交互；同时MEF让应用Program与扩展Program之间不产生直接的依赖，这样也允许在多个具有相同的扩展需求之间共享扩展Program。MEF的核心包括一个catalog（目录）和一个CompositionContainer（组合容器）。

* 1. MVVM模式

View的责任是定义屏幕上的结构和外观，在完美的情况下，view的code behind只包含一个调用InitializeComponent的构造函数，但是通常情况下，code behind会包含XAML标签难以表达的UI逻辑代码，所谓UI逻辑代码是指跟显示行为相关而不包含任何其它逻辑的代码，例如：复杂动画或者直接操纵view中可视化元素的代码。

View是一个可视化元素，例如window、page、user control、view定义了包含在其中的控件以及这些控件的布局和样式。在MVVM模式中，view通过DataContext属性引用它的view model，view通过数据绑定与它的数据上下文交互， vview中的控件绑定到view model中的property和command。同时通过change notification events通知视图其状态的变化。

View Model类

View model定义presentation logic和view所需要的数据，它不引用view，View model的责任是协调view和所需model的交互，通常情况下，view model和model是1对n的关系，view model可以选择将model类直接暴露给view类，使得view中的控件能直接绑定到model，在这种情况下，model类需要被设计成支持数据绑定和相应的通知机制。

Model封装业务逻辑和数据。它代表应用Program的领域模型在客户端的表现，Model封装业务逻辑和数据，它负责管理应用Program数据和封装业务逻辑和验证逻辑保证数据的一致性。model不直接引用view和view model。model类通常提供属性或者集合change notification events，方便和view的绑定。model类通常使用IDataErrorInfointerface和INotifyDataErrorInfointerface提供数据验证和错误报告。