FinanX 程序化金融交易系统客户端

体系结构描述文档

**FinanX Stocker程序化交易系统客户端体系结构文档更新历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人员 | 日期 | 变更原因 | 版本号 |
| 陈乾明 | 2014/08/12 | 文档创建,起草 | 1.0 |
| 陈乾明 | 2014/08/16 | 增加模块接口 | 1.1 |
| 陈乾明 | 2014/08/28 | 接口更正 | 1.2 |

目录

[1. 引言 4](#_Toc398411245)

[1.1 编制目的 4](#_Toc398411246)

[2. 体系结构模型 4](#_Toc398411247)

[2.1 逻辑视角 4](#_Toc398411248)

[2.2 程序启动进程 6](#_Toc398411249)

[2.3 开发包设计 6](#_Toc398411250)

[2.4 物理部署 9](#_Toc398411251)

[3. 接口视角 9](#_Toc398411252)

[3.1 Host Application模块 9](#_Toc398411253)

[3.2 Infrastructure模块 11](#_Toc398411254)

[3.3 Charts模块 12](#_Toc398411255)

[3.4 Login模块 13](#_Toc398411256)

[3.5 Market模块 14](#_Toc398411257)

[3.6 Navigator模块 15](#_Toc398411258)

[3.7 Statistics模块 15](#_Toc398411259)

[3.8 Trade模块 16](#_Toc398411260)

[3.9 Watch模块 16](#_Toc398411261)

[3.10 Wizard模块 17](#_Toc398411262)

[4. 信息视角 18](#_Toc398411263)

[5. 用户界面设计 20](#_Toc398411264)

[6. 系统体系结构设计思路 20](#_Toc398411265)

[6.1 C/S职责分配 20](#_Toc398411266)

[6.2 WPF 用户界面框架 21](#_Toc398411267)

[6.3 Prism组合界面框架 21](#_Toc398411268)

[6.4 MEF托管可扩展性框架 21](#_Toc398411269)

[6.5 MVVM模式 21](#_Toc398411270)

1. 引言
   1. 编制目的

本文档描述了对FinanX 程序化金融交易系统的概要设计，达到指导详细设计和开发目的，同时实现和测试人员的沟通。

本文档面向开发人员，测试人员及最终用户而编写，是了解系统的导航。

1. 体系结构模型
   1. 逻辑视角

Stocker采用WPF用户UI设计，使用Prism开发框架，选择了MVVM模式。

下图给出了Stocker在高层次上的逻辑体系：

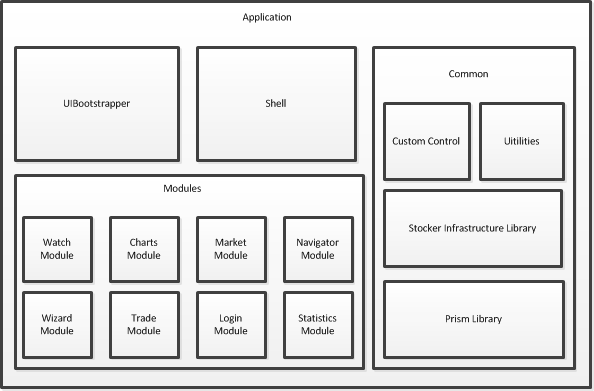


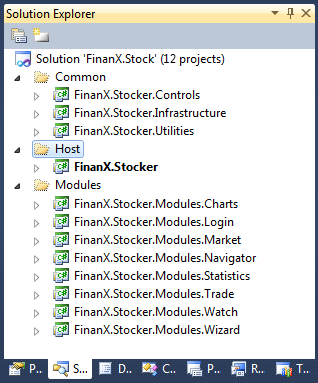
图 1 Finan-X Stocker Logic Achitecture

以下是对图1中主要元素的详细描述：

Application：Application是一个轻量级的容器，包含了用于组合各个功能模块的Shell外壳，还包含了一个UIBootstrapper启动器，用于设置MEF容器和初始化及加载功能模块。

Modules:程序的功能被分解成为8个具体的功能模块，每个模块可用于不同团队的开发。

* Watch Module:该模块包括了Watch List界面，用于查看用户关注的待购股票。
* Charts Module：该模块包括了Candle Stick, Volume Line, OHLC, Time Line报表，用于查看股市行情。
* Market Module：该模块包括了Market Summary股票市场概况列表，Stock Summary具体股票信息。
* Navigator Module：该模块包括了一个主页Index及程序功能导航栏。
* Wizard Module：该模块包含了应用程序设置界面。
* Trade Module：该模块主要负责获取股票数据，包括个股实时数据列表，股票历史数据，分类数据。
* Login Module：登录模块，包括了Login View登录界面及登录逻辑。
* Statistics Module：统计分析模块，该模块包括了计算股票购买组合界面，及分析股票收益信息逻辑。



* 1. 程序启动进程



图 2Application Process Step

程序启动进程

1. 程序使用UIBootstrapper启动，该Bootstrapper使用Mef容器初始化应用程序；
2. 程序初始化功能模块将使用到的Prism应用框架MefServiceLocatorAdapter；
3. UIBootstrapper创建并显示Shell View；
4. ModuleCatalog(Prism Library)查找需要加载的模块；
5. ModuleManager(Prism Library)加载并初始化各个功能模块；
6. RegionManager注入将模块界面region，并予以显示。
   1. 开发包设计

公共依赖库：

表 1公共依赖库

|  |
| --- |
| 公共依赖库 |
| Microsoft.CSharp,  Presentation.Core,  PresentationFramework,  System,  System.Core,  System.Data,  System.Data.DataSetEctensions,  System.Xml,  System.Xml.Linq |

表 2开发包设计

|  |  |  |
| --- | --- | --- |
|  | 名空间定义 | 依赖的其他库 |
| 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. 物理部署



图 3Deployment

1. 接口视角
   1. Host Application模块

Host模块的职责

|  |  |
| --- | --- |
| 组件 | 职责 |
| App | 主程序启动入口 |
| UIBootstrapper | UI启动器，用于设置MEF容器和初始化及加载功能模块 |
| Shell | 主窗口Shell外壳，标识Region注入区域 |
| ShellViewModel | Shell View的数据模型，用于存储显示在界面上的数据 |
| CallbackLogger | 程序日志记录器 |
| Splasher | 溅泻窗体控制器 |
| Splash View | 溅泻窗体 |
| DispatcherHelper | 为溅泻窗体传送消息工具 |
| MessageListener | 溅泻窗体消息监听器 |
| LoginShell | Login登录窗口的外壳 |

Host模块接口规范如表6所示。

表 3用户界面的接口规范

|  |  |  |
| --- | --- | --- |
|  | | |
| App. OnStartup | 语法 | protected override void OnStartup(StartupEventArgs e) |
| 前置条件 | 程序启动 |
| 后置条件 | 启动Splash，初始化设置，控制由AppDomain可能引发的异常 |
| UIBootstrapper. CreateShell | 语法 | protected override DependencyObject CreateShell() |
| 前置条件 | Shell View已加入MEF容器 |
| 后置条件 | 导出Shell View |
| UIBootstrapper. CreateLogger | 语法 | protected override ILoggerFacade CreateLogger() |
| 前置条件 | Logger实例已加入容器 |
| 后置条件 | 导出Logger |
| UIBootstrapper. ConfigureAggregateCatalog | 语法 | protected override void ConfigureAggregateCatalog() |
| 前置条件 | 已添加引用模块 |
| 后置条件 | 装配模块集 |
| UIBootstrapper.ConfigureDefaultRegionBehaviors | 语法 | protected override IRegionBehaviorFactory ConfigureDefaultRegionBehaviors() |
| 前置条件 | 需要注入View已使用Attributes标识 |
| 后置条件 | 创建自动注入工厂 |
| Splasher.ShowSplash | 语法 | public static void ShowSplash () |
| 前置条件 | 程序启动 |
| 后置条件 | 显示溅泻窗口 |
| Splasher. CloseSplash | 语法 | public static void CloseSplash () |
| 前置条件 | 程序初始化完毕 |
| 后置条件 | 关闭溅泻窗口 |
| MessageListener.ReceiveMessage | 语法 | public void ReceiveMessage ( string message ) |
| 前置条件 | Message属性已注册 |
| 后置条件 | 更新Message属性，并显示在Splash中 |
| DispatcherHelper. DoEvents | 语法 | public static void DoEvents ( ) |
| 前置条件 | 程序启动 |
| 后置条件 | 使用UI线程传送消息 |

* 1. Infrastructure模块

Infrastructure模块的职责

|  |  |
| --- | --- |
| 组件 | 职责 |
| 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 | 窗口标题信息接口 |

接口

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

* 1. Charts模块

Charts模块的职责

|  |  |
| --- | --- |
| 组件 | 职责 |
| CandleStickView | 蜡烛图（K线）视图 |
| CandleStickViewModel | 蜡烛图（K线）视图模型 |

接口

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

* 1. Login模块

Login模块的职责

|  |  |
| --- | --- |
| 组件 | 职责 |
| 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 |

接口

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

* 1. Market模块

Market模块的职责

|  |  |
| --- | --- |
| 组件 | 职责 |
| 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 |

接口

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

* 1. Navigator模块

Navigator模块的职责

|  |  |
| --- | --- |
| 组件 | 职责 |
| 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 |

接口

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

* 1. Statistics模块

Statistics模块的职责

|  |  |
| --- | --- |
| 组件 | 职责 |
| CompositionView | 股票组合计算控件 |
| CompositionViewModel | 股票组合计算控件模型 |

* 1. Trade模块

Trade模块的职责

|  |  |
| --- | --- |
| 组件 | 职责 |
| 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模块

Watch模块的职责

|  |  |
| --- | --- |
| 组件 | 职责 |
| 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 |

接口

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

* 1. Wizard模块

Wizard模块的职责

|  |  |
| --- | --- |
| 组件 | 职责 |
| WizardView | 设置窗口控件 |
| WizardViewModel | 设置窗口控件模型 |

1. 信息视角

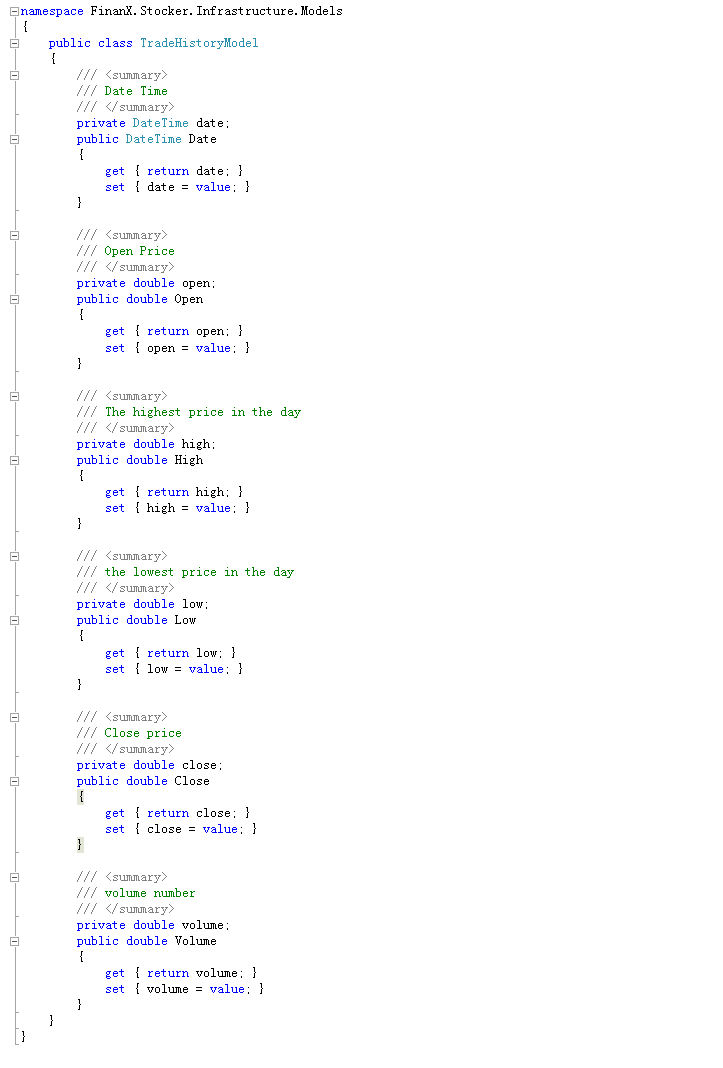


图 4Data Definition Example

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/S职责分配

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，从而美工和程序员可以各按其事，互不干扰。把界面拆分成若干小的模块（Module），从而使模块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。模块的拆分，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程序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程序交互；同时MEF让应用程序与扩展程序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程序。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。它代表应用程序的领域模型在客户端的表现，Model封装业务逻辑和数据，它负责管理应用程序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IDataErrorInfo接口和INotifyDataErrorInfo接口提供数据验证和错误报告。