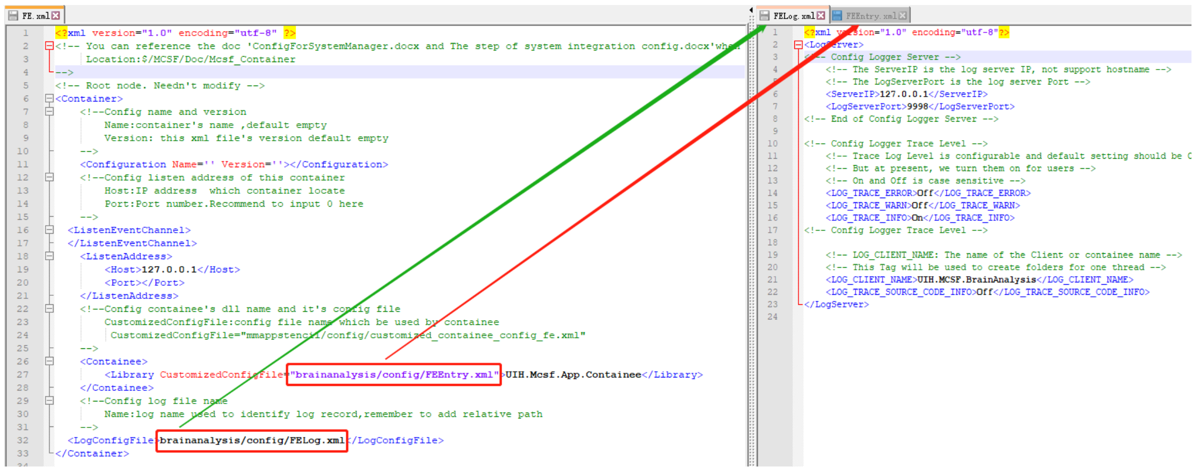
FE

依赖注入 Dependency Injection

把耦合从代码中移出去，放到统一的XML文件中，通过一个容器 `Container` 在需要的时候把这个依赖关系形成，即把需要的接口实现注入到需要它的类中。

层级

入口 `FE.xml`



FE.xml

```xml
<?xml version="1.0" encoding="utf-8" ?>
<!-- You can reference the doc 'ConfigForSystemManager.docx and The step of system integration config.docx'whe
     Location:$/MCSF/Doc/Mcsf_Container
-->
<!-- Root node. Needn't modify -->
<Container>
    <!--Config name and version
        Name:container's name ,default empty
        Version: this xml file's version default empty
    -->
    <Configuration Name='' Version=''></Configuration>
    <!--Config listen address of this container
        Host:IP address  which container locate
        Port:Port number.Recommend to input 0 here
    -->
  <ListenEventChannel>
  </ListenEventChannel>
    <ListenAddress>
        <Host>127.0.0.1</Host>
        <Port></Port>
    </ListenAddress>
    <!--Config containee's dll name and it's config file
        CustomizedConfigFile:config file name which be used by containee
         CustomizedConfigFile="mmappstencil/config/customized_containee_config_fe.xml"
    -->
    <Containee>
        <Library CustomizedConfigFile="brainanalysis/config/FEEntry.xml">UIH.Mcsf.App.Containee</Library>
    </Containee>
    <!--Config log file name
        Name:log name used to identify log record,remember to add relative path
    -->
  <LogConfigFile>brainanalysis/config/FELog.xml</LogConfigFile>
</Container>
```

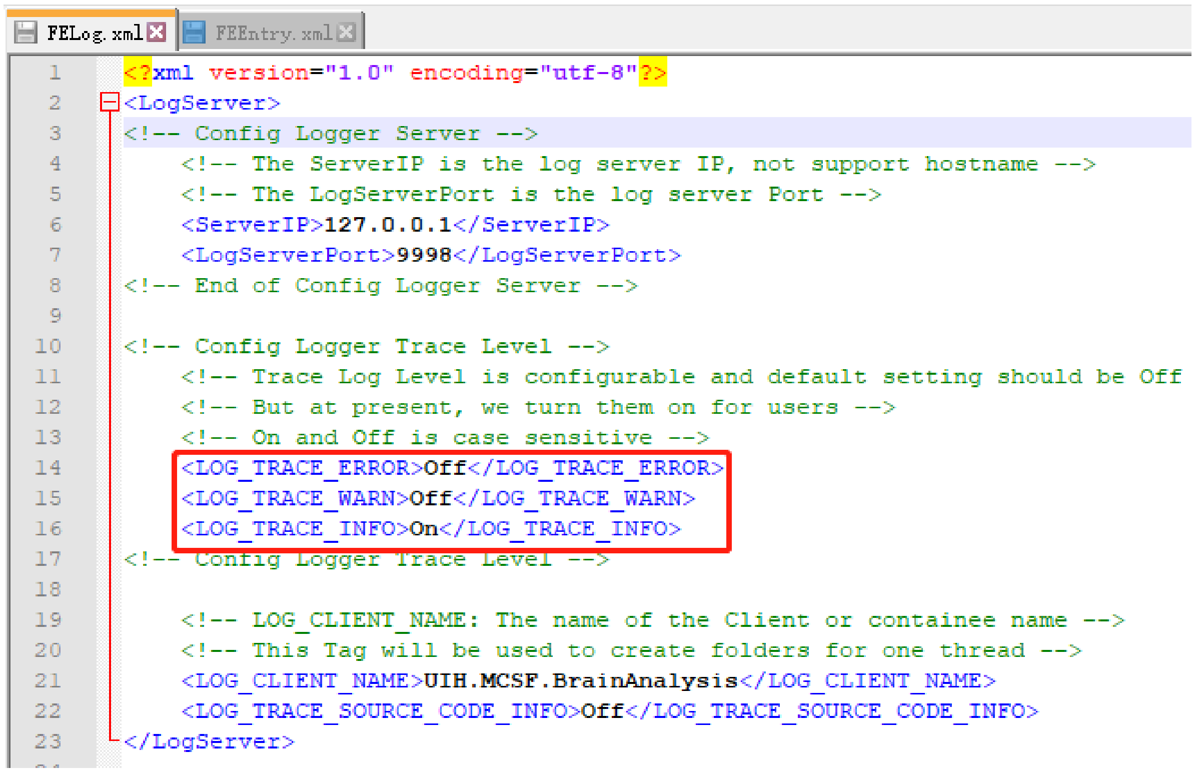
FELog.xml | FEEntry.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LogServer>
<!-- Config Logger Server -->
    <!-- The ServerIP is the log server IP, not support hostname -->
    <!-- The LogServerPort is the log server Port -->
    <ServerIP>127.0.0.1</ServerIP>
    <LogServerPort>9998</LogServerPort>
<!-- End of Config Logger Server -->

<!-- Config Logger Trace Level -->
    <!-- Trace Log Level is configurable and default setting should be C
    <!-- But at present, we turn them on for users -->
    <!-- On and Off is case sensitive -->
    <LOG_TRACE_ERROR>Off</LOG_TRACE_ERROR>
    <LOG_TRACE_WARN>Off</LOG_TRACE_WARN>
    <LOG_TRACE_INFO>On</LOG_TRACE_INFO>
<!-- Config Logger Trace Level -->

    <!-- LOG_CLIENT_NAME: The name of the Client or containee name -->
    <!-- This Tag will be used to create folders for one thread -->
    <LOG_CLIENT_NAME>UIH.MCSF.BrainAnalysis</LOG_CLIENT_NAME>
    <LOG_TRACE_SOURCE_CODE_INFO>Off</LOG_TRACE_SOURCE_CODE_INFO>
</LogServer>
```

调试的 `log` 输出设置



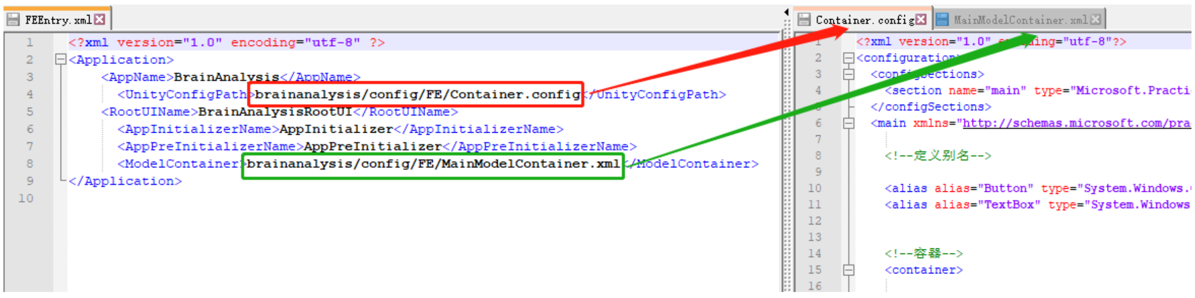
FELog.xml | FEEntry.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LogServer>
<!-- Config Logger Server -->
    <!-- The ServerIP is the log server IP, not support hostname -->
    <!-- The LogServerPort is the log server Port -->
    <ServerIP>127.0.0.1</ServerIP>
    <LogServerPort>9998</LogServerPort>
<!-- End of Config Logger Server -->

<!-- Config Logger Trace Level -->
    <!-- Trace Log Level is configurable and default setting should be Off
    <!-- But at present, we turn them on for users -->
    <!-- On and Off is case sensitive -->
    <LOG_TRACE_ERROR>Off</LOG_TRACE_ERROR>
    <LOG_TRACE_WARN>Off</LOG_TRACE_WARN>
    <LOG_TRACE_INFO>On</LOG_TRACE_INFO>
<!-- Config Logger Trace Level -->

    <!-- LOG_CLIENT_NAME: The name of the Client or containee name -->
    <!-- This Tag will be used to create folders for one thread -->
    <LOG_CLIENT_NAME>UIH.MCSF.BrainAnalysis</LOG_CLIENT_NAME>
    <LOG_TRACE_SOURCE_CODE_INFO>Off</LOG_TRACE_SOURCE_CODE_INFO>
</LogServer>
```

配置进入`Container.config`和`MainModelContainer.xml`



FEEntry.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Application>
    <AppName>BrainAnalysis</AppName>
      <UnityConfigPath>brainanalysis/config/FE/Container.config</UnityConfigPath>
    <RootUIName>BrainAnalysisRootUI</RootUIName>
      <AppInitializerName>AppInitializer</AppInitializerName>
      <AppPreInitializerName>AppPreInitializer</AppPreInitializerName>
      <ModelContainer>brainanalysis/config/FE/MainModelContainer.xml</ModelContainer>
</Application>
```

Container.config　MainModelContainer.xml

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="main" type="Microsoft.Practi
  </configSections>
  <main xmlns="http://schemas.microsoft.com/pra

    <!--定义别名-->

    <alias alias="Button" type="System.Windows.
    <alias alias="TextBox" type="System.Windows


    <!--容器-->
    <container>

```

`Container`：完成应用配置然后传给`Common`

`ModelContainer`：接口对应的实现部分

AppPreInitializer

传入应用名称、当前通信节点、所用UI资源等内容

```
public interface IAppPreInitializer
{
    void Initialize(string appName, FrameworkElement rootUI, ICommunicationProxy
proxy);
}
```

AppInitializer

进行一些基本初始化，包括对`CommunicationModuleModel`、`AppCommandHandlerModel`、`AppEventHandlerModel`的初始化以及注册`Handler`等

```
public interface IAppInitializer
{
    void Initialize(IModelContainer container, string appName, FrameworkElement
rootUI, ICommunicationProxy proxy);
}
```

Container.config

`Macrosoft Unity`：一个轻量级AOP框架，提供构造注入、拦截注入、属性注入、方法注入。

`<container>`标签中注册应用的资源

资源类型：

- Models
- Workflow, Workstep

- ViewModels
    - Command ViewModels
    - `Save` Command ViewModels
- Panel Operation
- Cell Operation
- Cell Initialize Item
- Cell Control Creator

MainModelContainer.xml

```
<Root>
    <Models>
    </Models>

    <ViewModels>
    </ViewModels>
</Root>
```

ModelItem

配置文件中键值所对应的属性

```
public class ModelItem
{
    public string Name;
    public string MapToClassName;
    public string Parameters;
    public string Path;
    public bool Keep;
    public string CascadeItems;
}
```

`MapToClassName`：`Container.Config`内对应的类名

AppModelBase

```
public class AppModelBase: IAppModel
{
    public ModelItem ConfigInfo { get; set; }
    public IModelContainer Container { get; set; }
    public virtual void Initialize() { }
}
```

`ConfigInfo`

`Container`

AppViewModelBase

```
public class AppViewModelBase : AppModelBase, IAppViewModel
{
    ...
    protected void RaisePropertyChanged(string propertyName){...}
    ...
}
```

继承自`APPModelBase`，比`APPModelBase`多一个`RaisePropertyChanged`方法

Models

RootUIModel

功能：管理`UI`的`Binding`和更新`CanExecute`状态等

ResourceModel

所有`Resource`都放在`ResourceDictionary`里，在`AppInitializer`初始化时，根据`RootUIName`拿到应用`MainControl`的`View`内容。

Other

`UnityModel`：`AppModelBase`

`DispatcherModel`：`AppModelBase`

`ProxyModel`：`AppModelBase`

配置 `UI` 资源：

```
<Item Name="UIResourceModel" MapToClassName="UIResourceModel" Keep="true"
Path="brainanalysis/config/FE/UIResource.xml"/>
```

配置 `快捷键` 绑定：

```
<Item Name="InputBindingModel" MapToClassName="InputBindingModel" Keep="false"
Path="brainanalysis/config/FE/InputBinding.xml"/>
```

其他主要配置：

Name	Class	File Name
`AppCommandHandlerModel`	`AppCommandHandlerModel`	CommandHanlder.xml
`AppEventHandlerModel`	`AppEventHandlerModel`	EventHandler.xml
`AllFunction`	`ControlAssemblyViewModel`	AllFunction.xml
`WorkStep1`	`ControlAssemblyViewModel`	WorkStep1.xml
`WorkStep2`	`ControlAssemblyViewModel`	WorkStep2.xml
`GeneralFunction`	`ControlAssemblyViewModel`	GeneralFunction.xml
`ExitFunction`	`ControlAssemblyViewModel`	ExitFunction.xml
`CommonTools`	`ControlAssemblyViewModel`	CommonTools.xml
`TissueROIControlTools`	`ControlAssemblyViewModel`	TissueROIControlTools.xml
`TissueROIDrawTools`	`ControlAssemblyViewModel`	TissueROIDrawTools.xml

ControlAssemblyViewModel

是`CommonTools`、`WindowLevelContextMenu`、`VRContextMenu`、`MPRContextMenu`等`Common`控件对应的类，有`IsEnabled`、`IsVisible`、`Children`和`Control`几种属性，以及`SetChildrenIsEnabled`、`Initialize`、`FindChildren`方法

ControlViewModel

主要是一些框架根据配置文件创建UI对象所需要的属性和方法，

有`CommandParameter`、`Container`、`ControlConfigInfo`、`BasicSettings`、`Control`、`Command`、`UI`、`Children`、`SelectedItem`、`Parent`、`IsInSilence`属性，

以及`EnterSilence`、`LeaveSilence`、`FindChild`和`Initialize`方法。

UIResource.xml

UIResourceModel

ExecuteItem

操作按钮，如`Button`、`RadioButton`、`CheckBox`等

```
public class ExcuteItemUI
{
    public string Name;
    public string Content;
    public string ToolTip;
    public string CheckedContent;
    public string UncheckedContent;
    public string ContentType;
    public string BitmapStretch;
    public string Width;
    public string Height;
    public string Margin;
    public string Tag;
    public string Style;
    public string DataTemplate;
    public bool IsVisible;
    public bool IsEnabled;
}
```

MenuItem

右键菜单

```
public class MenuItemUI
{
    public string Name;
    public string Header;
    public bool IsCheckble;
    public bool AllowSwitchToUncheckedWhenClicked;
    public bool IsAutoClosed;
    public bool RecognizesAccessKey;
    public string ContentType;
    public string DataTemplate;
    public bool IsVisible;
    public bool IsEnabled;
}
```

InputBinding.xml

```
<Item InputGeasture="Ctrl+S" Command="InteractivelySaveImageCommandViewModel"/>
<Item InputGeasture="F12" Command="PresetWindowing" CommandParameter="Default"/>
```

```
public class InputBindingItem
{
    public string Command;
    public string CommandParameter;
    public string CommandTarget;
    public string InputGeasture;
}
```

CommandViewModel

```
CanExecute()
OnExecute()
Execution
```

StateCommandViewModel

如果是在多种状态之间切换，则继承`StateCommandViewModel`

`StateCommandViewModel`派生自`CommandViewModel`，有一个`CurrentStates`属性，和调用事件`StateChanged`的`OnStateChanged`方法。

Workflow

接口：

`IAppWorkflow`：以 `MedViewerControl`为中心，管理一系列工作步骤、切换布局以及管理`Cell`

`IAppLayoutSwitcher` ：布局切换、双击放大、替换Cell类型

`IAppWorkStep` ：Workflow下的工作步骤

`ICellInitializer`：Cell添加图元、控件、四角信息等操作

类:

`AppCell`：添加图元，控件，不同的`Cell`设置不同的`Action`

IAppLayoutSwitcher

```
public interface IAppLayoutSwitcher
{

}
```