SCSF其实是基于CAB和企业库的一个软件工程开发工具包，核心就是CAB

1. Composite UI Application Block CAB

优点：

1. 允许构建由各个具有协作关系的独立模块组合成的复杂应用
2. 分离关注点，能见各个模块的开发的关注点和Shell的开发分离开来
3. 提供了一个能支持高质量的桌面应用开发的框架
4. 提高了生产力和节约了开发时间，进一步巩固了架构师和开发人员的劳动成果
5. 概念和术语

应用程序架构元素

* CAB：Composite UI Application Block的缩写
* Module：应用程序的组成部件，其中包含SmartParts，支持Service，业务逻辑和配置信息等
* ObjectBuilder：通过策略和配置信息自动创建对象实例的对象构造器
* Shell：承载用户接口元素，SmartPart，服务的外壳宿主程序
* Visualizer：CAB中提供的，可以查看应用程序中的WorkItem的动态分级视图的工具

外壳元素

* Shell application：承载了用户接口元素，SmartPart，服务的外壳宿主程序
* SmartPart：一个展现数据的视图比如控件，Windows窗体或是一个向导页
* SmartPartInfo：存放SmartPart的相关信息类，以便被workspace所使用，比如可以在Smartpartinfo中设置SmartPart的显示标题
* UIElement：一个以Shell作为宿主的控件，该控件能被多个Module所共享，这样的控件有：Toolbar button，menuitem，status panel等
* UIElement adapter：管理特殊类型的UIElement的显示的类
* Workspace：封装了控件和SmartParts的某种可视化的布局的组件，比如以Tab方式显示页面

相关模式

* Blackboard：通过提供一个共享信息的地方，使其他组件能设置或获取这个地方的信息，已达到信息共享的模式。
* Builder/Inversion of Control/Dependency Injection：该模式通过组件之间的依赖关系进行运行时的注入，来达到组件重用和松散耦合的目的。
* Event Broker：允许松散耦合的组件通过发布订阅的方式进行通讯的模式
* Memento：持久化一个对象的内部状态，并在需要的时候可以在后期对状态进行恢复的模式
* Model-View-Controller(MVC)：该模式将领域模型，视图和基于用户输入产生的行为(控制器)分成3个不同的组成部分。视图部分提供给用户进行交互，交互信息通过视图传到控制器，控制器更新模型，模型引发事件，从而更新视图。
* Model-View-Presenter(MVP)：该模式将领域模型，视图和基于用户输入产生的行为(控制器)分成3个不同的组成部分。视图部分提供给用户进行交互，交互信息通过视图传到控制器，控制器更新模型，模型触发事件到控制器，同时，控制器负责更新视图。

相关编程模型

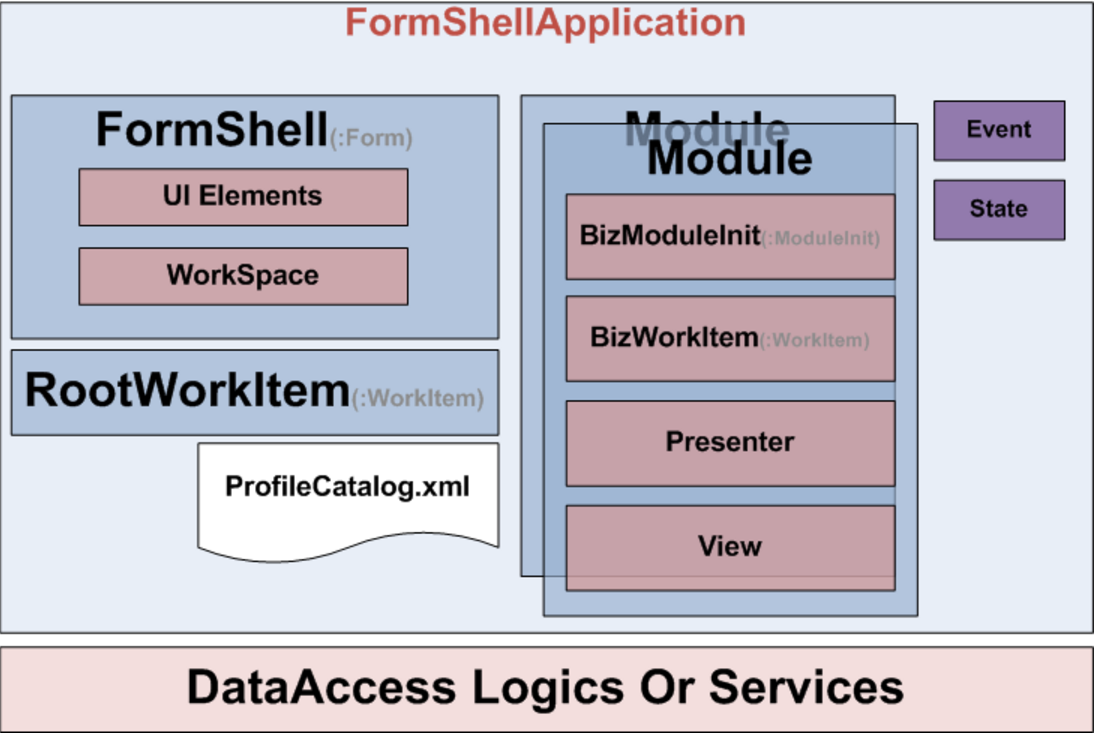
* Component：应用程序的可视或非可视组成部件，比如SmartParts，Services，和控件
* Container：包含了组件或服务的类
* Event broker：支持松散耦合的发布订阅事件机制的系统
* State：存在于WorkItem中，以键值的字典方式来存放共享信息。
* Service：以松散耦合的方式为其他的组件提供功能的组件。比如ModuleLoaderService
* WorkItem：运行时的组件和服务的容器，该容器能协作其中封装的用例代码的执行。一般WorkItem和用例对应。

相关角色

* Infrastructure developer：负责进行应用程序的基础服务的开发的开发人员
* Module developer：负责进行应用程序的业务组件的咖啡的开发人员
* Shell developer：负责建立应用程序外壳的开发人员
* SmartPart developer：负责开发应用程序的SmartParts的开发人员

1. Composite UI Application Block结构介绍

非常灵活的编程框架，将一个应用分离成不同的模块进行开发



首先一个CAB应用需要通过继承自FormShellApplication的应用程序类来进行启动。FormShellApplication需要传入两个类型参数，继承自WorkItem的类，和继承自Form的类(应用程序主界面FormShell)。主界面上可以放置供所有界面视图使用的公共的UI Element(如：Toolbar)。和显示用户界面的Workspace。WorkItem是封装了用例实现的容器，容器中的对象可以共享信息。WorkItem也可以包含下级WorkItem。

由于CAB的优点之一是能够很好的支持模块化开发，业务开发人员可以专注于某一方面业务模块的开发，如：仓库管理系统中，入库和出库就是同一个系统中的两个业务点，在CAB的支持下，完全可以将这两个业务点交由不同的开发人员进行开发，只要按照同样的既定的规范开发(界面规范，接口规范等)，就能很好地进行集成。CAB中通过Module很好的实现了这一点。

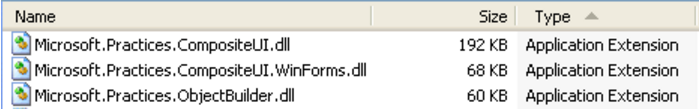
一般我们将module实现于dll文件中，每个dll文件可能包含系统某一方面的功能，当我们在独立开发好各个module之后，我们可以利用CAB有选择的加载这些功能模块，从而支持系统运行。要实现Module被加载很简单，只需要在ProfileCatalog.xml中将需要被加载的module配置进去就可以了。当前前提是被加载的模块需要符合CAB的Module设计的标准。如图中所示，这个dll中需要有一个继承自ModuleInit的类，Module被加载的时候，该类的Load方法将被调用，所以大家也可以在继承类中通过重载Load方法来扩展其加载时的行为。

一般情况下，在独立的模块中，我们还需要实现相应的继承WorkItem的类来实现其业务用例的封装，这个WorkItem我们需要将其添加到RootWorkItem的WorkItems集合中，以便和其建立联系。

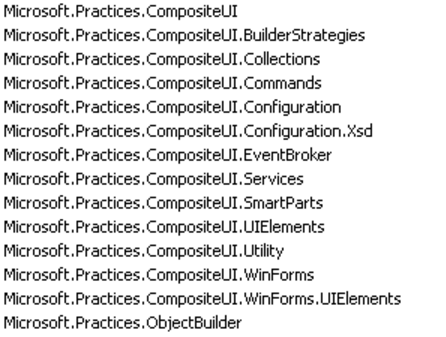
在WorkItem中可以使用MVP模式来实现我们的系统。上图中的View是在主窗体的Workspace中显示的和用户进行交互的界面，Presenter则是响应界面操作，处理业务逻辑的地方，Model这是我们的数据。

虽然我们的系统可以模块化的独立开发，各个模块之间实现了松耦合，但是系统运行时，CAB还是需要通过某种方式将各个模块糅合在一起，以便形成一个有机的横踢。首先CAB是一个IOC的容器，它可以在运行时根据需要实例化各种对象，并将其注入到对其有需要的对象中，达到对象的组装，然后可以通过发布订阅事件系统及贡献State实现了对象间的通讯

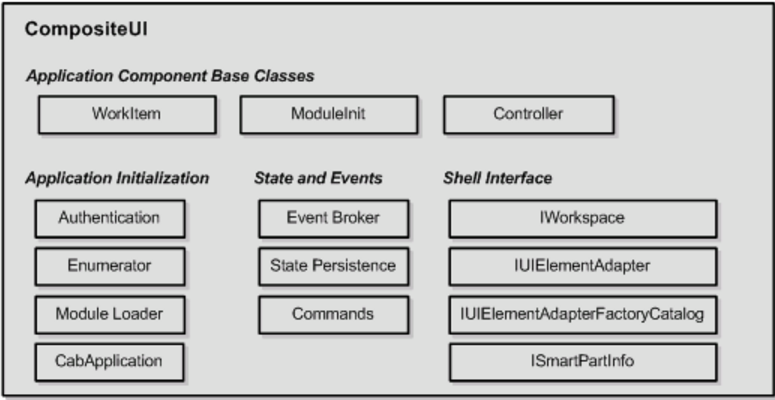
CAB涉及的DLL有：

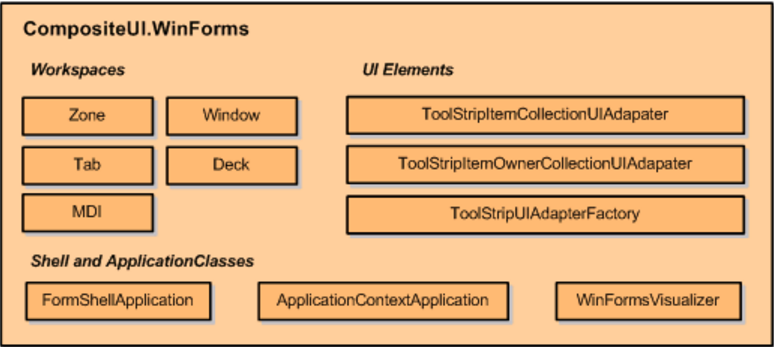


常用的命名空间如下：



附两张CAB中命名空间Microsoft.Practices.CompositeUI和Microsoft.Practices.CompositeUI.Winforms涉及的类和接口：





1. Composite UI Application Block(CAB) FormShellApplication介绍

现在我们来看看基于CAB的应用程序中非常重要的一个类。这个类可以认为是一个CAB应用的启动点。它就是FormShellApplication

FormShellApplication的继承关系如下：



该类需要传入两个类型参数，一个是继承自WorkItem的类(如果不需要通过重载WorkItem的OnRunStarted方法来实现更多处理，这里可以直接使用WorkItem)，一个是继承自Form的窗体。

Public class Program:FormShellApplication<MyWorkItem, ShellForm>

通常我们在Main方法中调用其Run方法，执行Run方法会初始化许多应用程序信息。可以通过override来重写或增加FormShellApplication的方法处理。

FormShellApplication的初始化包括(可以通过子类重载FormShellApplication的相关方法来判断其执行顺序)以下步骤：

1. RegisterUnhandleExceptionHandler
2. 创建Build strategies
3. 创建顶级WorkItem
4. 创建和初始化Visualizer
5. 添加Services
6. 创建Shell

CAB提供的标准服务如下，开发人员可以在此基础上进行扩展：

* SimpleWorkItemActivationService
* FileCatalogModuleEnumerator
* WindowsPrincipalAuthenticationService
* ModuleLoaderService
* DataProtectionCryptograohyService
* TraceSourceCatalogService
* CommandAdapterMapService
* WorkItemExtensionService
* WorkItemTypeCatalohService
* ControlActivationService

执行了FormShellApplication的子类的Run方法，一个CAB应用程序就算是启动了，随后就算根据自己的需要来执行WorkItem了。

1. Composite UI Application WorkSpace介绍

通过前面的介绍我们可以知道在静态Main方法中执行继承自FormShellApplication的对象实例可以进行很多框架的初始化工作，如加载模块，加载服务等。从FormShellApplication的类型定义中：

Public abstract class FormShellApplication<TWorkItem,TShell> ：WindowsFormsApplication<TWorkItem,TShell> where TWorkItem ：WorkItem where TShell : Form

可以看到，需要两个类型的参数，一个是WorkItem，一个是Form，FormShellApplication在进行初始化工作的时候会对这两个类型进行实例化，并且会将Form类型的实例Show处理。这里的Form类型的实例即是我们应用程序的主界面。

主界面架构是用户与系统的主要交互区域。它将作为一个容器，承载其他的业务界面。

CAB中提供了WorkSpace组件，它作为用户控件和SmartPart控件的显示容器，可以以各种各样的统一的显示方法呈现业务界面。也就是说WorkSpace可以以不同的风格呈现其中的用户控件。WorkSpace支持显示，隐藏，激活和关闭其中的用户控件。当前，CAB目前提供了几种默认的显示风格，接下来我们将一个一个介绍，如果开发人员觉得这些显示风格还不够用，当然也可以自己进行扩展。

所有类型的WorkSpace都是实现了接口IWorkSpace的。

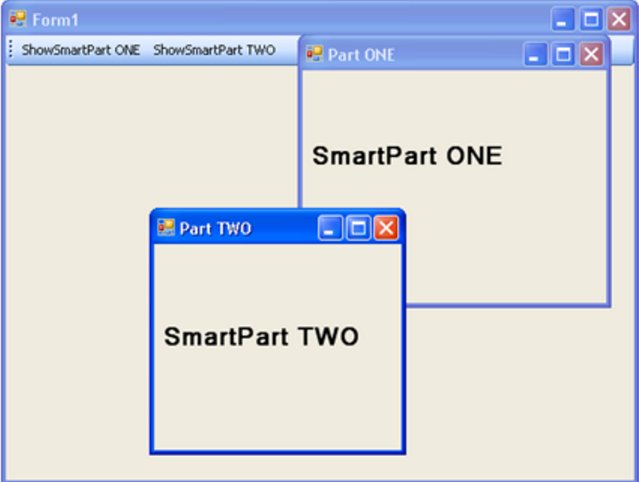
目前CAB提供的WorkSpace有：

* WindowWorkspace
* MdiWorkspace
* TabWorkspace
* DeckWorkspace
* ZoneWorkspace

下面来说说每种WorkSpace的特点：

1. WindowWorkSpace

该WorkSpace能将你需要在其中显示的用户控件在一个WinForm中显示出来，我们可以通过和该Workspace对应的WindowSmartPartInfo来设置用户控件的显示属性，如标题信息，是否模式窗口等。



1. MdiWorkspace

该Workspace是在WindowWorkSpace的基础上发展而来的，他同样将一个用户控件在单独的Fom中进行显示，并且，它还将以MDI的形式显示和管理其子窗体。对应用户控件的显示信息的设置是通过设置SmartPartInfo，然后作为参数传入Workspace的show方法

1. TabWorkspace

Tab页

1. DeckWorkspace

该Workspace将以类似重叠的卡片的形式来显示用户控件，当前激活的界面将在卡片的最上方。卡片的数序是由Wrkspace进行管理的，没有和该Workspace对应的SmartPartinfo

1. ZoneWorkspace

该Workspace将以平铺的方式显示用户控件，比如类似Outlook的界面，可以将一个界面划分成多个Zone，每个Zone都是作为呈现用户控件的一个容器。同样的，我们可以通过设置ZoneSmartPartInfo来设置用户控件在Workspace中的显示属性

6. WorkItem介绍

前面已经说到，CAB框架中除了提供了强大的支持功能，如依赖注入，事件机制，界面组合显示。。。还有一个专门封装业务逻辑的WorkItem。企业应用系统其实主要就是企业业务的自动化实现，一个应用框架提供的支持功能都只是为了更好地为业务服务。下面就来看看CAB中的WorkItem是如何封装业务用例的。

首先看看WorkItem都包含哪些元素？



从WorkItme的属性列表中我们可以看出，WorkItem中支持命令和事件，同时WorkItem可以嵌套，通过Parent进行关联，通过RootWorkItem可以获取顶层WorkItem，开发人员可以利用这一特性来组织自己也为用例和划分也为用例的粒度。WorkItem中使用State来共享信息，在同一个WorkItem容器中的对象可以共享访问这个信息。WorkItem容器中的对象对其中的Service都可以访问。

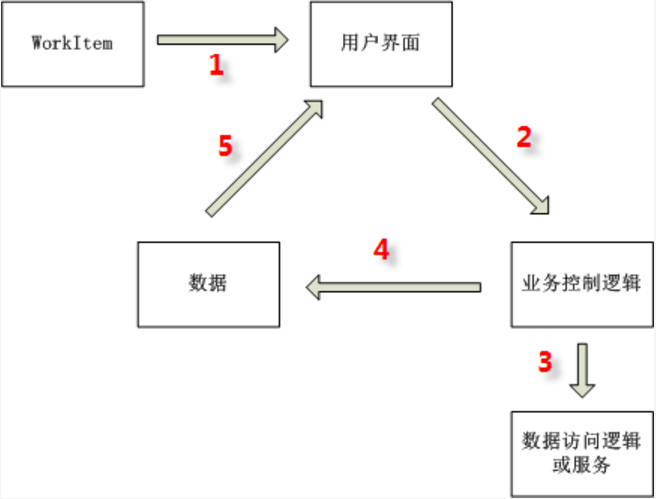
WorkItem通过调用Run方法进行启动，调用这个方法的时候会调用其他的OnRunStarted方法，一般我们自定义了一个WorkItem，如果有需要可以重载其OnRunStarted方法来自定义其启动逻辑。Run方法调用后会触发RunStarted事件。

一般我们会借助WorkItem提供的特性，采用MVC的模式进行业务用例的封装。使用SmartPart作为用户交互的UI显示部分(View)，创建一个控制类来进行业务逻辑的封装(Controller)，然后将业务数据存放于内存实体中(Model)。用户界面和内存实体采用绑定的方式关联起来。

MVC模式的应用，我们来看看他的初始化和用户交互过程中的操作是如何进行的？

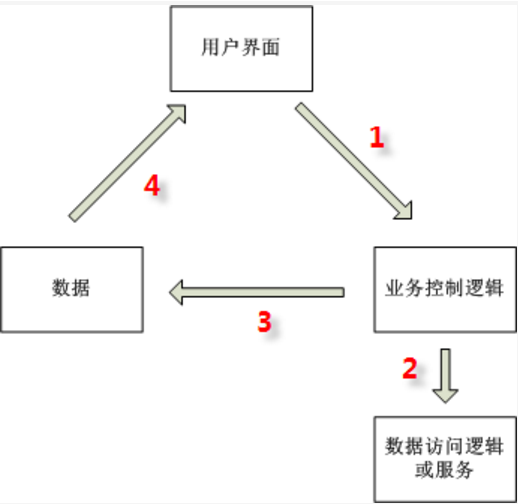
初始化：

1. 应用系统加载WorkItem，WorkItem将用户界面在WorkSpace中进行显示
2. 用户界面加载，调用控制逻辑进行初始数据的处理和获取
3. 控制逻辑获取初始数据
4. 控制逻辑将数据存放于内存实体中
5. 控制逻辑将数据与用户界面进行绑定，界面会自动根据绑定的数据源进行初始化显示



用户交互：

1. 用户操作用户界面，调用相应控制逻辑
2. 控制逻辑进行业务处理，进行相关服务或数据库访问
3. 控制逻辑修改内存实体中的数据
4. 由于数据与界面绑定，数据的变化直接反应到界面显示



1. CompositeUI Application Block 之 Smart Part

简单概念介绍

CAB提供了一个开放环境能很好的隐藏复杂度和提高生产力，通过高度抽象和关注点的分离，开发人员能够关注于业务逻辑提高基础框架代码的复用。Smart Part是整个CAB体系中重要的一部分，它可以将界面独立于业务逻辑，让界面和业务逻辑松散耦合起来。Smart Part的应用中有几个重要概念：

WorkSpace：作为一个容器，它可以统一的添加和显示视图。CAB提供了一组控件作为视图的容器，包括DeckWorkspace，MDIWorkspace，TabWorkspace，WindowWorSpace，ZoneWorkSpace。它们的作用有点类似普通Windows控件Panel，MDI窗口，TAB控件，Window窗体。在程序总根据名称可以通过WorkItem的WorkSpace集合索引到它们。

Smart Part：也可以被称作View，实际上是一个个自定义的控件。

其他像WorkItem，State之类的概念我在前两次的学习笔记中曾经介绍过了，这里就不再啰嗦了。

我们利用CAB的Workspace和SmartPart到底可以做些什么呢？WorkSPace可以一致的显示多个控件，Shell开发人员创建定制的Workspace以提供控件周围相似的框架和修饰，共享的布局和定位，炫目的界面切换。下面我们还是通过一个实例来逐步理解CAB之SmartPArt应用给我们带来的好处，特别 在多个数据来源，将多个界面块组合成一个统一美观的整体时的应用方便性。