**EC-Central项目**

**解决方案与工程Guideline**

Benny.T.Yang

2012-3-30

# 一、 总述

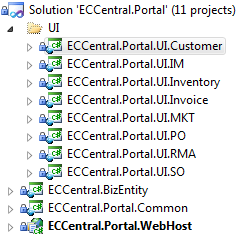
## （一）文档编写目的

本文档的目的是用于指导开发人员了解各个项目工程的细节结构、注意事项、文件规范等方面，对于要在项目中添加修改删除相应文件前，请先务必阅读本文档中对该项目的描述说明和注意事项。

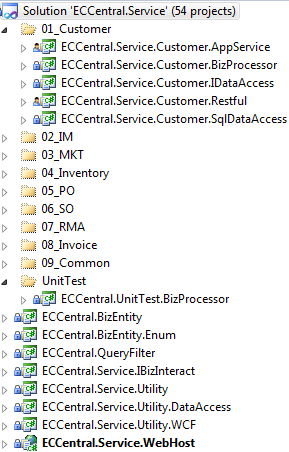
## （二）整体情况

整个EC-Central项目有分为2个解决方法，一个是Service端的ECCentral.Service，一个是UI Portal端的ECCentral.Portal；

* UI Portal端总共12个Project，其中4个是公共工程，另外8个是每个Domain一个自己的UI Portal工层：



* Service端总共54个Project，其中9个公共工程，另外是9个Domain每一个都有5个工程，总共为5 \* 9 + 9 = 54个工程；



# 二、Service端

## （一）基础公共工程

### 1. ECCentral.BizEntity.Enum工程

* **概述**

用来存放所有的枚举类型以及服务端的const常量，不会定义非枚举和非常量的其他类型；同时还会存放枚举所对应的现实文本的多语言资源文件；

* **工程类型**

Windows Class Library

* **依赖关系**

不依赖任何其他工程，会被除了ECCentral.Service.Utility、ECCentral.Service.Utility.DataAccess、ECCentral.Service.Utility.WCF、ECCentral.Service.WebHost外所有其他业务相关工程引用；

* **目录结构规则**

只有根目录，仅有一个存放资源文件的下级目录Resources，不允许在该工程中添加其他下级目录；

* **文件和类命名规则**
* 每个Domain对应2个文件，一个用来定义该Domain下的所有的枚举，一个用来定义该Domain下的所有的常量；加上Common Domain总共9个Domain，所以该目录下只能有18个cs文件，不允许再添加其他文件；  
  定义枚举的文件名为[Domain简名].cs，文件中的枚举类型命名根据业务需要自定义枚举类型名称；   
  定义常量的文件名为[Domain简名]Const.cs，文件中定义的静态类名和文件名相同，常量都定义在该静态类里，使用public修饰符；
* UI上需要使用到的枚举所对应的多语言文本内容的资源文件，都以Res开头，Enum结尾，每个Domain有且仅对应一个资源文件，文件命名规则：Res[Domain简名]Enum.resx，如：ResCustomerEnum.resx；
* **类命名空间规则**
* 除了资源文件自动生成的类型以为，其他类型使用ECCentral.BizEntity.[ Domain简名]，所以每个Domain对应的枚举或常量的cs文件都有一个自己Domain的namespace（同一个Domain的枚举和常量静态类的namespace相同）；
* 资源文件自动生成的类型的命名空间保持不变；
* **注意事项**

无

### 2. ECCentral.BizEntity工程

* **概述**

用来存放所有的业务模型实体的数据，将会是整个系统的最基础的工程；按照领域模型对业务进行分析和切割划分，该工程里的Entity类型都将对应到领域模型的数据部分，所以该工程里的Entity类并不是设计为数据传输用的实体（DTO），也不是设计为映射数据库用的数据库数据实体（DBEntity），而是纯粹根据业务模型来设计的。（但我们在整个系统里，也会利用该工程里的Entity作为DTO或者和DB的数据做映射，但这些做法只是为了节省在定义DTO和DBEntity以及它们之间做数据转化的工作量，而不是该工程里Entity的设计初衷和目的）

* **工程类型**

Windows Class Library

* **依赖关系**

仅依赖于定义枚举的ECCentral.BizEntity.Enum工程，会被除了ECCentral.QueryFilter、ECCentral.Service.Utility、ECCentral.Service.Utility.DataAccess、ECCentral.Service.Utility.WCF、ECCentral.Service.WebHost、ECCentral.Service.WebHost外所有其他的工程引用；

* **目录结构规则**

按照Domain来划分目录，每个Domain在项目根目录中建立一个对于目录，以Domain简名作为目录名，每个Domain的BizEntity都放在各自的目录里。每个Domain可以根据自己的情况，在自己的目录里根据业务模块再建立下级目录来存放对应的BizEntity代码。但无论Domain自己目录下面划分了多少级的目录，该Domain的所有BizEntity类型的命名空间都应该是统一的：ECCentral.BizEntity.[DomainBriefName] ；

* **文件和类命名规则**

按业务模型的英文名称来命名，不需任何的后缀和前缀，需要保持文件名和类名一致；

* **类命名空间规则**

统一使用ECCentral.BizEntity.[DomainBriefName]，无论Domain自己目录里有再划分多少级下级目录；

* **注意事项**
* 该工程里会有3个基础类型：  
  BaseInterface： 里面包含了5个最基础的接口，用于做实体的业务约束；  
  BizException ： 表示服务端的业务异常，通过构造函数传入异常文本，其属性Message为业务异常内容；

LanguageContent： 一个基础数据载体类型，派生自接口ILanguage，没有具体业务含义，用来装载指定语言的文本数据；其仅有2个字段：(1)Content字段表示具体文本内容；(2)LanguageCode字段表示文本内容所用语言的标准编码；使用场景一般是在某个BizEntity中，只有某个文本字段需要支持多语言，那么可以直接使用LanguageContent作为该属性类型，避免反复定义基本无业务意义的新类型；

### 3. ECCentral.QueryFilter工程

* **概述**

用来存放所有和业务无关的Query查询方法需要用到的查询条件实体类，该工程是一个基础工程；

* **工程类型**

Windows Class Library

* **依赖关系**

依赖于ECCentral.BizEntity.Enum工程，会被所有Domain的Restful、IDataAccess和DataAccess工程依赖；

* **目录结构规则**

按照Domain来划分目录，每个Domain在项目根目录中建立一个对于目录，以Domain简名作为目录名，每个Domain的QueryFilter类都放在各自的目录里。每个Domain可以根据自己的情况，在自己的目录里根据业务模块再建立下级目录来存放对应的BizEntity代码。但无论Domain自己目录下面划分了多少级的目录，该Domain的所有QueryFilter类型的命名空间都应该是统一的：ECCentral.QueryFilter.[DomainBriefName] ；

* **文件和类命名规则**

按相关查询的业务英文名称加上QueryFilter的后缀来命名，注意必须加上QueryFilter后缀，另外需要保持文件名和类名一致；

* **类命名空间规则**

统一使用ECCentral.QueryFilter.[DomainBriefName]，无论Domain自己目录里有再划分多少级下级目录；

* **注意事项**

该工程里会有1个基础数据类型：  
ECCentral.QueryFilter.Common.PagingInfo： 用来封装查询时的分页、排序信息；所有Domain在查询时需要分页时，都统一使用该数据类型来装载分页、排序的条件信息，而不要再定义其他类型了；

### 4. ECCentral.Service.IBizInteract工程

* **概述**

存放所有的各个Domain之间交互操作的接口，用于隔离各个Domain之间的直接依赖；

* **工程类型**

Windows Class Library

* **依赖关系**

该工程只依赖于ECCentral.BizEntity、ECCentral.BizEntity.Enum和ECCentral.Service.Utility（可能需要CodeNamePair类型）工程，会被所有Domain的AppService和BizProcessor工程所依赖；AppService里会实现该工程里的接口，而BizProcessor里是调用该工程里的接口；

* **目录结构规则**

没有下级目录，所有文件都放在工程根目录上，每个Domain就一个接口文件；

* **文件和类命名规则**

提供接口服务的Domain的简名加上BizInteract的后缀来做为名字；注意必须加上BizInteract的后缀，另外仍然需要保持文件名和类名一致；

* **类命名空间规则**

统一使用ECCentral.Service.IBizInteract；

* **注意事项**

该工程里会有1个基础数据类型：  
ECCentral.QueryFilter.Common.PagingInfo： 用来封装查询时的分页、排序信息；所有Domain在查询时需要分页时，都统一使用该数据类型来装载分页、排序的条件信息，而不要再定义其他类型了；

### 5. ECCentral.Service.Utility工程

* **概述**

存放各自业务无关的基础工具模块，现在包括了：实体和DataReader\DataRow的数据映射工具、高效反射执行工具、类型实例的依赖注入工厂、服务端多语言文本数据的帮助工具，以及其他一些工具类；

* **工程类型**

Windows Class Library

* **依赖关系**

不依赖任何其他工程，但会被除了纯接口（IDataAccess、IBizInteract）、纯数据实体（BizEntity、QueryFilter）和ECCentral.Service.WebHost以外的所有工程所依赖

* **目录结构规则**

按照工具模块的规模，如果就一个类实现了工具功能，那么就直接将该类放到工程的根目录下即可；如果需要多个类文件一起来完成某个工具功能，那么就需要在工程根目录下建立一个相关名称的目录，将该工具功能相关的类文件都放在该目录下，目录里面可以根据需要再建立下级目录；统一使用ECCentral.Service.Utility作为该工程里所有类的命名空间，无该类是否放在下级子目录里的；

* **文件和类命名规则**

没有明确的命名规则，名称尽量体现本身功能作用即可，但需要保持文件名和类名一致；

* **类命名空间规则**

无论是否在子目录里，都统一使用ECCentral.Service.Utility；

* **注意事项**

无；具体工具类的使用，请参照其他相关文档；

### 6. ECCentral.Service.Utility.DataAccess工程

* **概述**

提供针对数据库操作的工具类；

* **工程类型**

Windows Class Library

* **依赖关系**

该工程需要依赖ECCentral.Service.Utility；该工程会被所有Domain的DataAccess工程所依赖；

* **目录结构规则**

无，请不要擅自在该工程里添加目录和文件；

* **文件和类命名规则**

无，请不要擅自在该工程里添加目录和文件；

* **类命名空间规则**

使用ECCentral.Service.Utility.DataAccess及ECCentral.Service.Utility.DataAccess.[下级目录名]；

* **注意事项**

无；具体工具类的使用，请参照其他相关文档

### 7. ECCentral.Service.Utility.WCF工程

* **概述**

提供针对WCF Restful的相关类；

* **工程类型**

Windows Class Library

* **依赖关系**

该工程需要依赖ECCentral.Service.Utility；该工程会被所有Domain的Restful工程所依赖，也会被ECCentral.Service.WebHost所依赖；

* **目录结构规则**

无，请不要擅自在该工程里添加目录和文件；

* **文件和类命名规则**

无，请不要擅自在该工程里添加目录和文件；

* **类命名空间规则**

无论是否在子目录里，都统一使用ECCentral.Service.Utility.WCF；

* **注意事项**

该工程里会有2个基础数据类型：  
ECCentral.Service.Utility.WCF.QueryResult、QueryResultList： 所有的和业务无关的Query，在Restful这一层的方法返回给客户端的都需要包装为该类型对象实例。QueryResult类型就2个属性，一个DataTable，是本次查询所获取出的具体数据，一个是int型的TotalCount，表示数据库中符合该查询条件的所有记录总数（分页时会用到，不分页时该int型属性无用）。QueryResultList则是QueryResult的List，用来一次返回多个查询记录集的；

其他一些该工程的工具类的使用，请参照其他相关文档

### 8. ECCentral.Service.WebHost工程

* **概述**

服务端WCF Restful Service的宿主，是一个基于ASP.Net的Web工程；

* **工程类型**

ASP.NET Web Application

* **依赖关系**

该工程仅会依赖于ECCentral.Service.Utility；该工程不会被任何工程所依赖；

* **目录结构规则**

1. **Configuration目录**，里面放置Service端的所有的配置文件；  
   Configuration目录里的子目录设计规则为：

* **AppSettings**：这个是用来替代web.config的appSettings节点的。因为我们每个Domain的应用配置不允许直接写到web.config里，所以在这个目录下，为每个Domain都建立了一个[Domain简名].config的配置文件（没有多语言支持）；
* **CodeNamePairs**：这里都是放的每个domain的键值对列表的配置（按Domain划分目录，支持多语言）；
* **Data**：用来存放所有Domain的SQL配置的，在Data目录下按照Domain划分，每个Domain一个目录存放自己的SQL 配置文件；
* **MailTemplates**：用来存放邮件模板的，在其下有个Templates目录，所有Domain的所有邮件模板都定义在其下；
* **MessageResources**：用来存放业务异常信息的文本内容，支持多语言；在该目录下，按照Domain划分，每个Domain一个目录存放自己的异常信息配置；
* **PrintTemplates**：用来存放打印模板的，支持多语言，在该目录下，按照Domain划分，每个Domain一个目录存放自己的打印模板；
* **SegmentDict**：分词用的配置数据和字典数据；

1. **Download目录**，用来放置可以下载的、固定的文件，比如一些数据导入模板等；在Download目录下按照Domain名字为每个Domain建立一个子目录，然后每个Domain的文件都放在自己的目录下即可；
2. **Javascript目录和Style目录**，都是用来存放打印模板需要用到的一些web资源，比如js脚本、图片、css文件等；js脚本都放在javascript目录里，css文件放在Style目录里，图片则放在Style下的Images目录里；

* **文件和类命名规则**

1. **Configuration目录**下的各个子目录内配置文件的命名规则，如下：

* **AppSettings**：每个Domain对应一个文件，文件命名规则为[Domain简名].config；
* **CodeNamePairs**：每个Domain的每种语言对应一个文件，文件命名规则为[Domain简名] ].[语言代码].config；
* **Data**：每个Domain一个目录，文件名没有具体规则，尽量体现相关业务或操作，以config作为文件后缀名；
* **MailTemplates**：以Domain简名作为文件名前缀，以config作为文件后缀名：[Domain简名]\_[自定义名称].config。
* **MessageResources**：每个Domain一个目录，文件名规则为：[自定义名字].[语言代码].xml，以xml作为文件后缀名。
* **PrintTemplates**：每个Domain一个目录，文件名规则为：[自定义名字].[语言代码].[后缀名]，文件后缀名根据实际使用的报表来决定。
* **SegmentDict**：没有要求，根据需要命名即可；

1. **Download目录、Javascript目录和Style目录**下都根据需要来为文件命名即可，没有固定规定；

* **类命名空间规则**

无；

* **注意事项**

无；

### 9. ECCentral.UnitTest.BizProcessor工程

* **概述**

服务端的单元测试工程，主要是针对BizProcessor的单元测试，采用的NUnit框架；

* **工程类型**

Windows Class Library

* **依赖关系**

该工程会依赖于ECCentral.Service.Utility、ECCentral.BizEntity、ECCentral.BizEntity.Enum和每个Domain的AppService、BizProcessor以及SqlDataAccess；该工程不会被任何工程所依赖；

* **目录结构规则**

每个Domain一个单独的目录，存放自己Domain的测试代码；Domain在自己的目录里可以根据需要再建立下级目录；

* **文件和类命名规则**

建议的文件及类的命名规则为”[被测试的Processor对象名]Test”，如：RefundRequestProcessorTest；

* **类命名空间规则**

创建类文件时，VS.Net自动根据目录结构生成的命名空间即可；

* **注意事项**

该工程会有个编译后命令，用于将WebHost工程下的Configuration里的配置文件都拷贝到单元测试的程序集所在目录，以便测试代码的正常运行；

## （二）Domain相关工程

### 1. ECCentral.Service.[Domain简名].Restful工程

* **概述**

每个Domain存放自己向UI Portal端所暴露的WCF Restful Service类，以及交互需要用到的Message数据实体类；  
这里面的WCF Service没有使用接口，而都是用的具体类型，每个类型上都必须加上[ServiceContract]的标签，类型的public方法上都需要加上[WebInvoke]或[WebGet]的标签；

* **工程类型**

Windows Class Libraray

* **依赖关系**

该工程会依赖于ECCentral.BizEntity、ECCentral.BizEntity.Enum、ECCentral.QueryFilter、ECCentral.Service.Utility、ECCentral.Service.Utility.WCF以及自己Domain的AppService、IDataAccess工程；该工程不会被任何工程所依赖；（但为了运行调试方便，会被ECCentral.Service.WebHost引用，以便在编译运行WebHost的时候能自动将其dll文件都copy到WebHost的bin目录里）

* **目录结构规则**
* 所有的Restful Service类型都直接放在工程的根目录下；
* 而对于和UI Portal端交互用的Message数据实体，则是在该工程根目录下建立2个子目录，分别为：  
  RequestMsg：用于存放方法参数上的Message实体（即从UI Portal发送到Service端的数据载体）  
  ResponseMsg：用于存放方法返回的Message实体（即从Service端返回给UI Portal的数据载体）
* **文件和类命名规则**
* 对于Restful Service，所有的Class名称都需要以Service关键字结尾：[自定义名]Service，如：CustomerService，需要保证文件名必须和类名一致（如果有partial class，则保证文件名主体名和类名一致）；
* 对于RequestMsg目录下的类，需要以Request作为名字后缀：[自定义名] Request，如：OrderCheckRequest，需要保证文件名必须和类名一致；
* 对于ResponseMsg目录下的类，需要以Response作为名字后缀：[自定义名] Response，如：CustomerVisitLogResponse，需要保证文件名必须和类名一致；
* **类命名空间规则**
* 工程根目录下的Restful Service类型都以ECCentral.Service.[Domain简名].Restful为命名空间；
* 对于RequestMsg目录下的类，都以ECCentral.Service.[Domain简名].Restful. RequestMsg为命名空间；
* 对于ResponseMsg目录下的类，都以ECCentral.Service.[Domain简名].Restful. ResponseMsg为命名空间；
* **注意事项**
* 该工程的Service里应该只涉及和分布式交互技术相关的处理，以及对Message和BizEntity的转换过程，而不应该出现任何和业务、应用相关的需求逻辑处理；
* 另外对于Message的定义和使用，是按需使用，即如果BizEntity即可满足Restful Service方法的入参或返回值要求，则尽量直接使用BizEntity来作为入参或返回值，而无需另外再定义Message；只有在没有适合的BizEntity来装载入参或返回值的数据时，才需要也必须要定义相应的Message来作为数据载体；
* 特别禁止为了满足Restful的数据传输要求，而去修改BizEntity的定义，去添加属性以满足分布式交互的数据传输要求，这种做法和行为是被坚决禁止的，因为我们的BizEntity反应的是业务本身的数据模型，而不是单纯的数据传输载体；
* 为了调试部署方便，所以该工程添加了编译后命令，在编译后会通过XCOPY命令把编译的结果都copy到Service端的WebHost的bin目录下；

### 2. ECCentral.Service.[Domain简名].AppService工程

* **概述**

每个Domain里用来实现IBizInteract接口和为Restful Service提供相应服务的；该工程的Service里面不会包含任何的核心业务相关的逻辑，而只包含和具体技术、具体应用相关的需求处理，比如Cache、日志监控、文件上传处理、文件解析、XML解析等，而真正的业务处理转交给下面的BizProcessor去做。

* **工程类型**

Windows Class Libraray

* **依赖关系**

该工程会依赖于ECCentral.BizEntity、ECCentral.BizEntity.Enum、ECCentral.Service.IBizInteract、ECCentral.Service.Utility以及自己Domain的BizProcessor工程；该工程会被自己Domain的Restful工程所依赖；

* **目录结构规则**

一般情况下，类文件放在根目录即可，如有需要可按模块建立下级目录；

* **文件和类命名规则**

所有类都必须以AppService结尾：[自定义名]AppService，如：CustomerPointsAppService；需要保证文件名必须和类名一致（如果有partial class，则保证文件名主体名和类名一致）；

* **类命名空间规则**

要求该工程下所有类型都以ECCentral.Service.[Domain简名].AppService作为命名空间，无论是否放在子目录里的；

* **注意事项**

该工程相当于Domain的Biz Core对外的服务接口， 一方面为UI Portal提供服务能力（透过Restful Service, Restful Service本身很薄，只负责分布式技术和消息封装转换相关的处理），另外一方面通过实现IBizInteract来为其他Domain提供服务能力；~~而在该工程内部可以是一个类的方法，即实现了IBizInteract接口方法，同时该方法又被Restful Service所调用。~~为了让IBizInteract 的实现更干净，我们确定在每个Domain的AppService中，统一增加一个 命名为：BizInteractAppService的类，这个类继承该Domain的的IBizInteract并提供实现；这样尽量让接口实现的AppService与自己Domain所使用的AppService分开。

### 3. ECCentral.Service.[Domain简名].BizProcessor工程

* **概述**

业务处理的主体，按照领域模型划分出来，该工程里的类型对应着自己Domain里的业务模型的业务行为。（Domain模型的数据都被分隔到ECCentral.BizEntity工程里了）

* **工程类型**

Windows Class Libraray

* **依赖关系**

该工程会依赖于ECCentral.BizEntity、ECCentral.BizEntity.Enum、ECCentral.Service.IBizInteract、ECCentral.Service.Utility以及自己Domain的IDataAccess工程；该工程会被自己Domain的AppService工程所依赖；

* **目录结构规则**

一般情况下，类文件放在根目录即可，如有需要可按模块建立下级目录；

* **文件和类命名规则**

所有类都必须以Processor结尾：[自定义名] Processor，如：CustomerPointsProcessor；需要保证文件名必须和类名一致（如果有partial class，则保证文件名主体名和类名一致）；

* **类命名空间规则**

要求该工程下所有类型都以ECCentral.Service.[Domain简名].BizProcessor作为命名空间，无论是否放在子目录里的；

* **注意事项**

该工程里需要做好业务的内聚性，尽量把业务本身相关的处理逻辑都放入该工程里；但需要注意区分应用逻辑和业务逻辑，应用逻辑相关的处理应该放到AppService里，真正的业务本身处理逻辑则应放在该工程里。

### 4. ECCentral.Service.[Domain简名].IDataAccess工程

* **概述**

每个Domain在该工程里定义了自己需要的数据访问接口；通过该工程能体现出上面的层次对数据持久化的需求，同时通过接口来隔离上面层次对具体持久化技术的依赖，以便后期切换其他数据持久化方式时（比如重构DB，或者切换DB，甚至换NoSQL方案），不用影响到上层调用者。

* **工程类型**

Windows Class Libraray

* **依赖关系**

该工程会依赖于ECCentral.BizEntity、ECCentral.BizEntity.Enum、ECCentral.QueryFilter、ECCentral.Service.Utility（可能需要CodeNamePair类型）；该工程会被自己Domain的Restful和BizProcessor工程所依赖；

* **目录结构规则**
* 对于业务相关的数据访问接口放在根目录；
* 对于和业务无关的数据查询接口，则放在NoBizQuery这个子目录 里
* **文件和类命名规则**

所有接口类型都必须以I开头，DA结尾：I[自定义名]DA，如：ICustomerDA；需要保证文件名必须和类名一致（如果有partial class，则保证文件名主体名和类名一致）；

* **类命名空间规则**
* 对于放在根目录的接口，以ECCentral.Service.[Domain简名].IDataAccess作为命名空间；
* 对于放在NoBizQuery这个子目录 里的接口，以ECCentral.Service.[Domain简名].IDataAccess.NoBizQuery作为命名空间；
* **注意事项**

无；

### 5. ECCentral.Service.[Domain简名].SqlDataAccess工程

* **概述**

存放是针对SQLServer数据库持久化的的实现类，是对自己Domain的IDataAccess里所有接口的实现；

* **工程类型**

Windows Class Libraray

* **依赖关系**

该工程会依赖于ECCentral.BizEntity、ECCentral.BizEntity.Enum、ECCentral.QueryFilter、ECCentral.Service.Utility、ECCentral.Service.Utility.DataAccess以及自己Domain的IDataAccess工程；  
该工程不会被任何工程所依赖，我们会通过依赖注入的方式将其注入到我们的整个系统里；（但为了运行调试方便，会让WebHost工程引用该工程，以便在编译该工程的时候能自动将其dll文件都copy到WebHost工程的bin目录）

* **目录结构规则**

一般情况下，类文件放在根目录即可，如有需要可按模块建立下级目录；

* **文件和类命名规则**

所有类都必须以DA结尾：[自定义名] DA，并且自定义名需要和所实现的IDataAccess里的接口自定义名一直，如：OrderCheckDA；需要保证文件名必须和类名一致（如果有partial class，则保证文件名主体名和类名一致）；

* **类命名空间规则**

要求该工程下所有类型都以ECCentral.Service.[Domain简名].BizProcessor作为命名空间，无论是否放在子目录里的；

* **注意事项**
* 统一使用ECCentral.Service.Utility.DataAccess里提供的持久化工具类来做，具体做法和API都类似于之前的IPP3的持久化方式。
* 每个Domain里统一都会建一个EnumCodeMapRegister的类型文件，用来注册所有的枚举和数据库字段值的映射关系。
* 为了调试部署方便，所以该工程添加了编译后命令，在编译后会通过XCOPY命令把编译的结果都copy到Service端的WebHost的bin目录下；

# 三、UI Portal端

## （一）基础公共工程

### 1. ECCentral.BizEntity.Enum工程

* **概述**

用来存放所有的枚举类型，将会是直接映射对应的Service端的BizEntity.Enum工程，采用文件Link的方式，直接把Service端BizEntity.Enum工程的代码加入该工程里；也会把Service端定义的Enum的对应的多语言显示文本的资源文件Link过来；

* **工程类型**

Silverlight Class Library

* **依赖关系**

不依赖任何其他工程，会被ECCentral.Portal.Common和ECCentral.Portal.UI下所有Domain的工程引用；

* **目录结构规则**
* 从Service端的BizEntity.Enum工程采用文件Link的方式加入进来的文件，都原样参照其在Service端BizEntity.Enum工程里的目录结构，都是放在工程根目录下；
* Portal端UI上需要使用到的枚举所对应的多语言文本内容的资源文件，都通过从Service端Link文件的方式放到该工程根目录下的Resources目录里；
* **文件和类命名规则**
* 从Service端的BizEntity.Enum工程采用文件Link的方式加入进来的文件其文件名保持不变；
* Link过来的资源文件的文件名、类名也保持和Service端一致；
* **类命名空间规则**
* 从Service端的BizEntity.Enum工程采用文件Link的方式加入进来的文件保持其原有命名空间不变；
* Link过来的资源文件的命名空间也保持不变；
* **注意事项**

请不要添加任何其他文件进入该工程

### 2. ECCentral.BizEntity工程

* **概述**

用来存放所有的业务模型实体的数据，将会是直接映射对应的Service端的BizEntity工程，采用文件Link的方式，直接把Service端BizEntity工程的代码加入该工程里；

* **工程类型**

Silverlight Class Library

* **依赖关系**

仅依赖于ECCentral.BizEntity.Enum工程，会被ECCentral.Portal.Common和ECCentral.Portal.UI下所有Domain的工程引用；

* **目录结构规则**

从Service端的BizEntity工程采用文件Link的方式加入进来的文件，都原样参照其在Service端BizEntity工程里的目录结构；

* **文件和类命名规则**

从Service端的BizEntity工程采用文件Link的方式加入进来的文件其文件名保持不变；

* **类命名空间规则**

无；

* **注意事项**
* 该工程下有个SerializableAttribute的类型，该类的主要作用是为了让BizEntity能正常编译通过，因为在Silverlight中已经没有的[Serializable]的attribute，但是Service端会用到，也会在BizEntity定义，所以为了让Link到Silverlight这边来的并且使用了[Serializable]的attribute 的BizEntity能顺利编译通过做的一个MockType，欺骗编译器而已；
* 请不要添加任何其他文件进入该工程；

### 3. ECCentral.Portal.Common工程

* **概述**
* 用来存放Portal UI上会公用的Silverlight的控件；
* Portal端UI上通用的数据格式转换，比如日期时间格式、货币格式等
* 业务无关的工具类，包括BizEntity和ViewModel的自动映射赋值工具、高效反射工具、RestClient（用于向Restful Service端发起http请求的）、dynamic对象和ViewModel的自动映射赋值工具、枚举转换工具、验证工具等；
* 一个专门用来存放Portal端需要用到的常量的类；
* **工程类型**

Silverlight Class Library

* **依赖关系**

会依赖于ECCentral.BizEntity、ECCentral.BizEntity.Enum，会被ECCentral.Portal.UI下所有Domain的工程引用；

* **目录结构规则**
* Portal 端UI上公用的Silverlight的控件、公用的QueryFilter实体、公用的Facade及ViewModel，都放到Components目录下存放；
  + QueryFilters目录，存放公共需要用到的查询QueryFilter文件，以Link的方式从Service端的Service.QueryFilter工程里引用添加进来；
  + Facades目录，存放公共需要用到的Facade，主要是和Service端的Common Domain交互的；
  + Models目录，存放公共需要用到的ViewModel，主要是配合公共Facade使用的；
  + UserControls目录，存放公用的Silverlight控件，其下可以按照每个控件一个目录的原则
* Portal端UI上通用的数据格式转换工具类都放到Converters目录下；
* 其他的业务无关的工具类都放到里Utilities目录下，根据实现工具类文件的数量，如果只有一个文件即可实现工具，那么直接放到Utilities目录下即可，如果需要多个文件才能实现该工具，那么可以在Utilities目录下建立相应子目录；
* 专门用来存放Portal端需要用到的常量的类直接放在工程根目录；
* **文件和类命名规则**
* 放在Components目录下的类：
  + QueryFilters目录，存放的以Link的方式从Service端的Service.QueryFilter工程里引用添加进来的，文件名称保持和Service端一致即可；
  + Facades目录里文件都以Facade结尾：[自定义名]Facade，如：WebChannelFacade；
  + Models目录里文件都以VM结尾：[自定义名] Model，如：WebChannelVM；
  + UserControls目录，存放公用的Silverlight控件，每个控件的目录里根据自己文件类的角色来命名，如Facade作用的就以Facade结尾，ViewModel作用的都以为VM结尾，Message和Filter则都Link方式引用Service端的文件的，保持文件名不变即可；
* 放在Converters目录下的类，无论有多少下级目录的，都统一使用命名空间ECCentral.Portal.Common.Converters；
* 放在Utilities目录下的类，无论有多少下级目录的，都统一使用命名空间ECCentral.Portal.Common.Utilities；
* 放在工程根目录的常量类，使用ECCentral.Portal.Common的命名空间；
* **类命名空间规则**
* 放在Components目录下的类，按照目录结构来生成命名空间；
* 放在Converters目录下的类，无论有多少下级目录的，都统一使用命名空间ECCentral.Portal.Common.Converters；
* 放在Utilities目录下的类，无论有多少下级目录的，都统一使用命名空间ECCentral.Portal.Common.Utilities；
* 放在工程根目录的常量类，使用ECCentral.Portal.Common的命名空间
* **注意事项**

在Components目录下的三个目录，需要注意下面规则：

* QueryFilters：存放的是多个控件或其它组件会公用到的QueryFilter，如果只是某个控件自己单独需要的QueryFilter，请以Link方式添加到UserControls目录下自己控件的子目录里；
* Facades：存放的是多个控件或其它组件会公用到的Facade类，如果只是某个控件自己单独需要的Facade，请定义在UserControls目录下自己控件的子目录里；
* Models：存放的是多个控件或其它组件会公用到的ViewModel类，如果只是某个控件自己单独需要的ViewModel，请定义在UserControls目录下自己控件的子目录里；

### 4. ECCentral.Portal.WebHost工程

* **概述**

Portal端Silverlight的的宿主，是一个基于ASP.Net的Web工程；

* **工程类型**

ASP.NET Web Application

* **依赖关系**

该工程不会依赖于解决方案中的任何工程，也不会被解决方案中任何工程所依赖；

* **目录结构规则**

主要是Oversea框架需要的一些目录和文件，暂时不用关心；

* **文件和类命名规则**

主要是Oversea框架需要的一些目录和文件，暂时不用关心；

* **类命名空间规则**

无；

* **注意事项**

调试客户端可以使用该工程，但最好先编译整个解决方案，然后再Debug启动运行改工程。

## （二）Domain相关工程

### 1. ECCentral.Portal.UI.[Domain简名]工程

* **概述**

每个Domain自己的Silverlight UI端；

* **工程类型**

Oversea Portal V5 （Silverlight Application ）

* **依赖关系**

该工程会依赖于ECCentral.BizEntity、ECCentral.BizEntity.Enum、ECCentral.Portal.Common，不会被其他工程引用

* **目录结构规则**

固定的5个目录：

* Facades：
  + Facade文件直接放在该目录下即可；
  + 对于和Restful Service那边交互需要用到的Request消息和查询的QueryFilter，都放在RequestMsg目录下，都通过添加文件Link的方式从Service端ECCentral.Service.[ 相关Domain].Restful和ECCentral.QueryFilter里按需引入过来；可以根据实际情况，在其下再建立相应的子目录；
  + 对于和Restful Service那边交互需要用到返回值的Response消息，则放在ResponseMsg目录下，也是通过添加文件Link的方式从Service端的ECCentral.Service.[相关Domain].Restful里按需引入进来；可以根据实际情况，在其下再建立相应的子目录；
* Models：存放自己Domain的所有的ViewModel，其下可以根据需要建立相应子目录；
* Resources：存放所有的UI端界面上文字信息的多语言的资源文件；
* UserControls：存放自己Domain所使用的Silverlight UserControl的View（主要都是一个大的Page的View切割成多个小的View），其下可以根据实际情况建立子目录；
* Views：存放自己Domain的所有的Silverlight功能页面文件，其下可以根据实际情况建立子目录；
* **文件和类命名规则**
* 对于Restful Service，所有的Class名称都需要以Service关键字结尾：[自定义名]Service，如：CustomerService，需要保证文件名必须和类名一致（如果有partial class，则保证文件名主体名和类名一致）；
* 对于RequestMsg目录下的类，需要以Request作为名字后缀：[自定义名] Request，如：OrderCheckRequest，需要保证文件名必须和类名一致；
* 对于ResponseMsg目录下的类，需要以Response作为名字后缀：[自定义名] Response，如：CustomerVisitLogResponse，需要保证文件名必须和类名一致；
* **类命名空间规则**
* 对于Link方式引入进来的文件类，都保持原本身的命名空间不变；
* 对于其他的类型，则直接按照其目录结构来确定命名空间（即VS.net自己默认的命名空间生成方式）；
* **注意事项**

该工程是整个系统的界面层，请保持没有任何业务逻辑，仅有UI交互逻辑（比如数据有效性验证、控件状态的控制等）即可；