**注：本文中说的强类型枚举enum就是特指的C#中的枚举类型；**

**一、使用理念：**

1. 贯穿整个系统的都应该用强类型枚举enum的枚举值，而不应该再出现其他的code；
2. 仅在DataAccess层才关心枚举类型enum的枚举值与其DB存储值之间的映射关系；
3. 仅在UI层才关系枚举类型enum的枚举值与其显示文本之间的映射关系；

**二、实现目的：**

1. 我们的系统的业务代码里，就只关心含有业务意义的强类型的枚举enum，而不关心具体的string或int的code值是多少，对开发人员来说针对enum的coding更为友好（有智能提示，不用再去记忆具体的code值），重构代码也更方便，业务语义的表达也更清晰；
2. 强类型的枚举enum与其具体将在DB中存储的值之间的映射关系只会在DataAccess层里，并且只会出现一次；这样系统这部分适应变化的能力就会更强
3. 强类型的枚举enum与其具体将在UI上显示的文本之间的映射关系只会在UI层里，并且只会出现一次；这样系统这部分适应变化的能力就会更强

**三、具体做法：**

1. BizEntity上需要为枚举的属性，都必须定义为强类型枚举enum，不允许使用string或int的code来表示；
2. 在DataAccess层中，在统一的一个地方定义一次强类型枚举enum与DB存储值之间的映射关系；然后在持久化相关的读或写方法中，支持自动做强类型枚举enum与DB存储值之间的相互转换，从而保证开发人员只用做一次强类型枚举enum与DB存储值之间的映射定义即可，其它时候都不用再关心它们的映射和转换；
3. 对于No Biz Query所查询到的DataTable里，如果有枚举相关的列，需要在DataAccess层指定相关列并完成转换（DataCommand会提供相应的ExecuteTable的重载方法，自动完成从db的存储值到enum类型值的mapping），这样数据到达上层调用者时，已经是具体的强类型枚举enum了，而不是int或string的code；
4. 对于UI上的Dropdownlist、Checkbox等之类的控件，在绑定枚举选项列表时都必须是直接绑定的强类型的枚举enum作为其选项的值，从而保证获取控件的值时直接拿到的就是强类型的枚举enum的值，而不是其它的int或string的code；
5. 在UI 上，在统一的一个地方定义一次强类型枚举enum与显示文本之间的映射关系（现在的做法是在enum类型上添加了一个description的attribute），然后在UI层提供了一个Converter来方便的在界面绑定时将强类型枚举enum的值直接转换为其对应语言的文本来显示；
6. 在Service端，需要生成打印或邮件的html内容时，可能需要用的枚举的显示文本信息，那么可以使用ECCentral.Service.Utility中的EnumHelper中定义的扩展方法public static string ToDisplayText(this Enum value) 来得到枚举对应的显示文本；
7. 针对历史遗留数据的处理，可能会有这样一种情况，针对某一个枚举类型，在之前有定义了一套对应的DB存储值，但后来做了一次系统重构改版后，将该枚举类型的DB存储值修改为另外一套了，这样导致现在一个枚举值，在DB里可能存在多个存储值（当前真正在用的只是一个，但是历史数据里表示该枚举值的可能还有其他存储值，也就是写入的时候enum还是只会对应唯一的存储值，但是从DB里load出来的时候，由于历史数据原因可能会有多个DB Code映射到同一个enum的值上）；针对这种情况的处理方式是：使用EnumCodeMapper类里的静态方法

public static void AddExtraCodeMap<TEnum>(Dictionary<int, object[]> maps)

public static void AddExtraCodeMap<TEnum>(TEnum value, params object[] codes)

该方法有2个重载形式，可以用来注册额外的1对多的映射关系（1个enum对应多个DB Code），这样对于前面提到的历史遗留数据的多个DB Code，都可以通过该方法注册。注意：该方法只会影响到从DB Load数据出来的时候会额外根据该映射关系来进行一次转换；而对于写入DB的时候，该方法注册的映射关系没有任何作用和影响；

EnumCodeMapper.AddExtraCodeMap的调用，建议还是放到DataAccess层中的EnumCodeMapRegister类里，和第2点的映射关系注册放在一起，便于管理；

1. 针对另外一种历史遗留问题：

对于同一个Enum枚举值，在Save到DB里的两个不同表的时候，会Save为两个不同的值（比如一个表里是int字段，而另一个表里是char字段，这个可能也是因为之前系统在不同的开发维护阶段做出了不同的设计导致的问题），显然之前的映射关系注册就不支持这种情况了，因为之前的针对写入DB的映射关系都是1对1的；

建议的做法是：

针对这种“特殊”的枚举，会和一套DB存储值用EnumCodeMapRegister做一套标准的映射关系，而针对另一张表的该枚举字段的存储值，则是在DA层持久化和该表相关枚举字段的时候，需要用switch case来做一次转换操作，将枚举值转换为对应的DB存储值。该switch case的转换操作都放在各自Domain的DA层一个名为LegacyEnumMapper的static class里，为该“特殊”枚举建一个static的方法，方法里来将枚举值转换为具体的DB存储值，方法名就为：ConvertXXXX(XXXX value) ----- XXXX就为该“特殊”枚举类型的名字，方法返回值根据需要设计即可；如果又再遇到一个类似的“特殊”枚举，那就直接在LegacyEnumMapper里只需要再增加相应的Convert方法即可了。