**请注意：本文中所说的业务中的枚举项并不是指具体的C#中的enum，而是单纯指业务中可能遇到的固定选项形式的情况；**

**一、对于业务枚举项的三种实现方式：**

在我们的系统里，会有很多使用枚举项的地方，比如SO的订单状态、Customer的等级、投诉责任原因等，我们有定义了3种做法：

1. **定义和使用C#中的枚举类型enum；**

* UI显示：
* 显示文本的存储：使用UI Portal端的resx资源文件来配置存放要显示的文本，有固定规范；
* 对多语言的支持：支持；
* coding代码难度：Portal端有相应的工具类支持，很方便；
* DB持久化：
* DB存储：持久化对应的char或int值；
* coding代码难度：很方便，在DataAccess端有相应的mapping工具的支持，支持自动的双向转换，转换过程可以对开发人员透明；

1. **使用服务端配置文件，用服务端的配置工具类读取出来后是个CodeNamePair类型；**

* UI显示：
* 显示文本的存储：使用Service端的xml文件来配置存放要显示的文本，有固定格式；
* 对多语言的支持：支持；
* coding代码难度：Portal端有相应的工具类支持，很方便；
* DB持久化：
* DB存储：持久化CodeNamePair的Code属性值；
* coding代码难度：很方便，在DataAccess端有相应的工具的支持，支持string和CodeNamePair对象之间的双向转换，转换过程可以对开发人员透明；

1. **在数据库中有相应的配置表**

对应的会在系统里定义相应的BizEntity，会和其他的BizEntity一样走AppService 🡪 BizProcessor 🡪 DataAccess的路线，这是最复杂的方式；

* UI显示：
* 显示文本的存储：存储在DB中，需开发人员自己设计表结构和相关字段，最灵活；
* 对多语言的支持：不一定，需要根据需求自行考虑设计；
* coding代码难度：完全自行设计AppService 🡪 BizProcessor 🡪 DataAccess的代码，最麻烦；
* DB持久化：
* DB存储：存储对应配置表中记录的主键（SysNo字段）
* coding代码难度：基本的DataAccess操作，开发人员完全自己控制sql入参或者sql返回字段与Entity属性的mapping

**二、对于三种方式选择使用的原则：**

我们在业务中遇到这种有枚举选项的地方，我们从上面3种做法中选哪一种呢？有个大致的原则和标准，如下：

1. 首先识别这个业务枚举选项**是否需要配置的UI界面功能**从而可以允许系统使用者自己去维护和配置该业务枚举选项，**或者是否前台需要公用配置的**，**如果是，那么请使用上面的第3种最复杂方式的做法**，比如我们系统中的ShipType；
2. 识别这个业务枚举选项，**是否在后期可能会发生变化并且在发生变化后并不会导致我们业务逻辑的变化**（增加、删除或修改），**如果是，那么请使用上面的第2种方式的做法**，比如仅给界面上显示用的投诉责任原因，在增加或减少选项后，我们的程序逻辑不会受任何影响，无需改动任何代码；
3. 其他情况下，都请使用上面的第1种方式做法，在C#中定义枚举类型来做
4. 另外，如果一个属性表示的就只有是和否的情况，比如IsXXX或者HasXXX这种属性，那么请直接使用bool?类型（BizEntity的属性如果为值类型，都请用对应的Nullable<>），而不要再去定义一个Yes or No的enum了