Sanjel应用软件设计原则及实现规范

Sanjel Application Design Principles and Implementation Guidelines

# 设计原则

# SOLID 原则

* 单一职责原则 Single Responsibility Principle
* 开放闭合原则 Opened Closed Principle
* 里氏替换原则 Liscov Substitution Principle
* 接口隔离原则 Interface Segregation Principle
* 依赖反转原则 Dependency Inversion Principle

# 模式原则

设计模式对于同类的相似问题提供了通用的解决方法。它所提供的是概念上的解决方法，对于具体问题往往还需要具体实现一份代码。利用[抽象类](https://zh.wikipedia.org/wiki/%E7%B1%BB_(%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6)" \o "类 (计算机科学))或[接口](https://zh.wikipedia.org/wiki/%E6%8E%A5%E5%8F%A3)往往可以在特定的设计模式中达到代码的复用。

# 实现规范

# 总则

* 复用原则 – 在着手实现子系统、服务或是复杂逻辑前，先寻找是否有已经存在的类似实现，避免无谓的重复劳动。不要重新定义轮子。
* 封装原则 – 如果一个实现被考虑用来复用，首先要将其封装成方法、类或服务，以实现接口隔离。
* 规范原则- 在同类技术实现时要遵从同样的规范，团队要定义这种规范，以保证代码的可维护性。规范可以改进和进化，进化路径要清晰并记录在案。

# ASP .Net MVC 实现规范

# Syncfusion控件使用规范

# 开发过程规范

* 在写代码前，需要设计。定义类也是设计的一部分，即使没有逻辑代码，也应该将类定义先提交。这样可以及时得到反馈，开始验证迭代。
* 在编写代码时，要先写接口，然后再实现。接口是设计的过程。不要整个文件地拷贝其他代码，引进不必要的方法。

# 测试规范

# 测试数据

测试数据命名时可以把name space考虑进去，否则名字会太长

耐心写测试数据，复杂的集成测试数据集应该由负责架构的人来完成，具体实现的测试数据集由开发员自己来完成。

维护测试数据时要小心维护Id,root\_id, parent\_id这些值，它们可能会在验证关系的时候用到。如果不维护，等到需要的时候，重加一遍几乎是不可能的事情。

测试数据的设计，对于引用master data或是有规范的数据，要与原始数据严格一致并遵守规范。

# 单元测试

单元测试和集成测试的测试数据集要分开

单元测试的precondition数据准备要直接并明确，不要带不必要的依赖

单元测试可以层次化设计，从最小方法测试，实体测试，一层一层向上集成，避免产生交叉依赖。

# 集成测试

# 测试的覆盖

1. 测试要覆盖的每个子项的结果

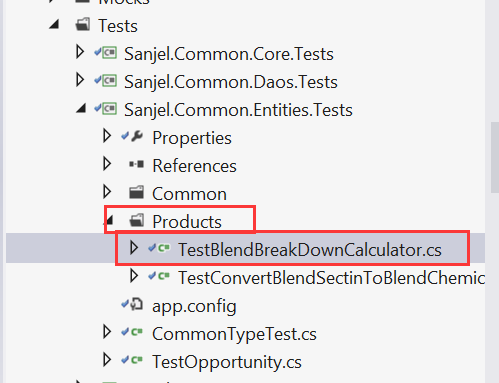
在TestBlendBreakDownCalculator中，每个测试只对表面的测试数据进行了检查，而对下面子项的结果没有检查。所以要有相应的测试。

# 测试程序存放位置

1. 对于不同目的的测试程序存放位置应遵循以下规则

对于实体计算方法的测试都放在其对应的实体的测试文件夹下

TestBlendBreakDownCalculator这个测试是对实体计算方法的测试，它不是前台逻辑，所以放在UIP Test的项目中是不合适的，应该放在Product



# 命名规范

# TestData命名规范

1. 见名知意

例如不能用TestData1、TestData2…命名

1. 如果这个测试数据本身的名字就可以区分开，则用它本身的名字加TestData命名

例如BlendChemical的测试数据：Create010A()，CreateLITEmix1325AB()

1. 如果测试数据本身的名字不能区分，则要参照这个测试数据与哪个实体相关，用这些实体的名字进行区分

例如ProductHaulTestData中，ProductHaul在创建时需要用到CallSheetNumber以及BaseBlend，那么可以考虑用CallSheetNumber以及BaseBlend的名字命名：CreateProductHaulForCallSheet 1074548SURFACEmixLWPRO()

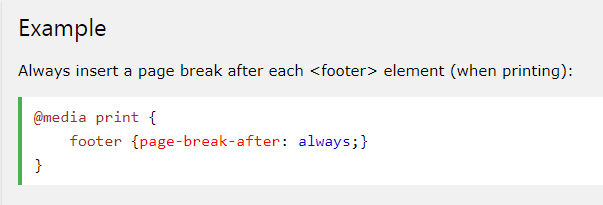
# 日常行为规范

# 确保代码持续工作

每次提交之后的代码必须是工作的，尤其是每天下班前最后一个人提交代码要特别注意，代码提交后必须在其他人电脑上更新确保代码是工作的。如客户发现代码不工作，责任人每次罚款50元人民币作为项目基金。

# 常用方法汇总

# 页面分页打印



# Eservice online架构规范

# 组件职责

* + 1. MVC 前端组件，按照ASP .Net MVC的架构要求，实现前端表示层。这一层尽可能薄，专注前台用户交互需求的实现。
       1. View: HTML, CSS， javascript script 加上syncfusion 控件库实现页面控制
       2. ViewModel： 我们定义ViewModel作为MVC中的M， 是为了解耦前端解决方案对业务数据模型的依赖，降低前端实现的难度。ViewModel与View是同构的，这样可以通过对ViewModel和Controller进行单元测试，以保证逻辑的正确性。
       3. Controller: ViewModel的数据准备和前端请求响应。只处理前台页面操作逻辑的处理。对于涉及到业务数据或状态改变的逻辑，不在此处理。如菜单准备，显示格式准备。
    2. eServiceWebContext的职责是为前端进行数据准备，这里是eServiceOnline与底层模型的数据接口，为Controller提供统一的数据源，避免前端重复写数据处理逻辑，如将ProductHaulLoad 转换成ProductHaulModel，供前端使用。它调用eService Services 或MicroService来读取后台数据。

它ViewModel到业务数据模型之间转换。包括以业务模型为语义基础的请求装配和返回结果的解释。

在当前实现中的BusinessProcess组件（目前包括RigBoardProcess，ProductHaulProcess），在架构层面并不应该是独立的一层，它只是将业务逻辑进行封装，不产生新的对象实例，实际上应该是静态类。这样做的好处是码清晰，职责明确，可维护性好。否则，eServiceWebContext将是个巨无霸，实现上现在已经1500多行代码了。

* + 1. Service 层
       1. EserviceService

eService Service是基于 Sanjel eService Framework的服务，它基于自动聚合框架实现聚合对象的组装与拆解，连接数据访问层(DAO)，实现聚合对象的数据访问。它是eService WinForm Client 与WCF 服务的基础。

* + - 1. EserviceMicroService

eService MircoService是直接与数据访问层连接，实现低粒度数据访问与操作的轻量服务，它是在eService Online中引进的一种机制。基于此我们可以实现低粒度的Web API用于应用扩展和集成。

TODO： 我们需要研发基于EServiceMicroService的聚合机制，以改善聚合实现代码分散的问题。

* + 1. DAO层与EService中的都 一样的，但是SQLite的处理，与SQL Server不一样。认

eServiceWebContext只是一层非常薄的数据访问层，用于online的前端与后端的交互，他并不包括复杂的业务逻辑，他存在必要性是因为我们可以更换不同的后端实现，或者基于后端适应不同的前端。

由于web应用对性能的要求比较高，所以在online的实现中并没有采用eservcie原有的具有组装聚合对象的Service服务，而是采用了轻量级的microservice，用以提高应用的性能，但在某些必要的情况下还是可能需要应用具有组装聚合对象的Service服务。

目前的实现中，我们对大对象的查找可以有三种方式，一种是通过原有service进行组装，实现数据的查找，这在online中是不建议的，因为它需要多次连接数据库，性能比较差，第二种是通过创建调用微服务的服务去组织大对象，还有一种是直接在服务中通过直接调用相应的Dao来组织大对象。

在接下来的实现中，由于对性能的迫切需求，我们可能去更改或扩展eservice 的dao，使它可以更灵活的获取我们需要的数据，而非全部取出，或者创建数据库的视图，把要查找的多张表的内容做成一个虚拟表以提高性能。

# EserviceExpress前台架构开发规范

# 组件职责

* + 1. MVC 前端组件，按照ASP .Net MVC的架构要求，实现前端表示层。
    2. eServiceWebContext的职责是为前端进行数据准备，这里是eServiceExpress与底层模型的数据接口，为Controller提供统一的数据源，避免前端重复写数据处理逻辑，如将ServiceTicket中的PriceLineItemSection 转换成PricingLineRecord，供前端使用。它调用eServiceExpressGateway来读取后台数据。
    3. EserviceExpressGateway的职责是协调两个不同数据源LocalDataManager和LocalSqliteDataManager。
    4. LocalDataManager和LocalSqliteDataManager的职责是实体数据整理，聚合，分割，状态管理等，它与eService中的Service层目的相同。它调用DAO层进行数据读取。
    5. DAO层与EService中的都 一样的，但是SQLite的处理，与SQL Server不一样。认真理解Olivia设计的架构，遵守使用规则。

# MVC规范Viewbag 和ViewData使用

在代码中有很多使用Viewbag或者 ViewData传递对象实例的使用，可以使用model尽量都使用model。因为用Viewbag或者 ViewData可以随意传递任何需要的信息，而不再考虑实体的结构，整体结构的层次，破坏设计的一致性。随着项目规模的增长，可维护性会大大降低。

# 对MVC理解的问题

在MVC架构中，ViewModel应该是与View是对应的，它不是传统的MVC中的设计，它的模式应该是MVVM和MVC的一种结合，引入ViewModel的目的在于把View和逻辑分离得更清晰，使提View可以在脱离Web界面的情况下也可以被测试。一般来讲，它是一个View专用的，不可以被复用。如果在设计上可以把控web控件的层次叠加，那么ViewModel也应该是继续和聚合的关系，还是一一对应的。

1. 将LocalJobs页面的ViewModel，改名为LocalJobViewModel
2. ViewModel的引用仅局限于MVC工程内，所有由eServiceWebContext提供的数据到ViewModel的数据转换，在Controller中完成。
3. 在后台业务类到ViewModel之间根据需要可以加入Model类，原则是将实现变得更简单，而不是刻意去做。在这种情况下，Model到ViewModel的数据转换在Controller中完成，而业务类到Model之间的转换在WebContext中去完成。

**此文档不是一成不变以，它在不断迭代中改进，有任务想法都可以提出来。**