**术语**  

Pascal case － 所有单词第一个字母大写，其它字母小写。

例：Person, OrderDetail, OilTank。 

Camel case － 除了第一个单词，所有单词第一个字母大写，其他字 母小写。

例：oilLevel, customerName

## 操作规范

#### 模板及格式化

* 开发人员必须保证代码格式化的一致性，否则可能会导致代码冲突， 轻微的耗费人力合并代码；严重时可能导致代码丢失，引起bug或者故障。
* 服务端格式化文件



* 客户端格式化文件



* 每次提交代码之前，必须对java代码format。

#### 代码提交

* 提交前，先从SVN 中更新代码，确保提交的代码没有错误和冲突。
* 提交时，如果发生冲突时，先看历史说明，再找相关人员确认，坚决不允许强制覆盖。
* 提交时，先在自己的研发环境中运行程序，确保程序运行正常、没有错误，才可以提交代码。

#### 垃圾清理

* 对从来没有用到的或者被注释的代码、配置文件等要坚决从系统中清理出去，避免造成过多垃圾。

#### 数据库修改

* 开发功能时，不允许随意修改UAT数据库中的数据，尤其是基础数据。如果有修改数据库的需求，应在SIT环境中修改测试。
* 在SIT数据库中的所有修改都应该放到升级脚本中，每次转测试版本前需确认升级脚本无误。
* 已经转测试的脚本不允许再修改，如果有变更，应该新增脚本。

## 日志规范

#### 日志输出

* 生产代码禁止以System及Throwable.printStackTrace的方式输 出日志信息，必须用Logger替代。
* 对info级别的日志输出，除非必要，一般情况下不要使用。
* 任何情况下都不允许由于记录日志错误导致业务处理失败。
* 对非info和error级别的日志输出，必须加上判断条件，不要无条件输出。

#### 错误日志

* 对于由于系统原因造成业务处理失败的事件，需要记录错误日志。非系统原因的业务处理失败，不应该记录错误日志（推荐使用warn级别），避免错误日志过大，影响紧急情况下的故障分析不诊断.
* 输出错误日志时，一定要输出日志描述，最好是简洁的英文字母，方便排查问题时快速搜索。

## 注释规范

* 所有的类、方法(set/get除外)、常量、dto类中的所有属性、判断语句都需要加上注释。
* 注释不必过多，在关键逻辑中添加注释即可。

## 异常处理

* 捕捉到的异常，不允许不做任何处理就截断，至少要输出日志，或重新抛出。
* 最外层的业务使用者，必须处理异常，将其转化为用户可以理解的内容。不允许将异常堆栈信息直接抛给用户。

## 安全规范

* 所有页面都应该加上防重复提交的功能。
* 严禁出现方便调试的后门页面/接口。
* 用户的敏感信息如密码、 加密串、 月结卡号、费用等不能泄露，否则可能会带来不安全因素。可能会导致敏感信息泄露的方式有:Logger、URL的get参数（因为URL的get参数会在apache日志中被输出） 。

## 命名规范

* 包名全部小写，不要出现下划线等特殊符号

例如：com.sf.module.clienttask.biz.impl

* 所有类命名使用 Pascal 表示方式，使用名词组合。

例如：UserManager, ClassLoader, HttpHeaderResult

* 接口命名使用字母“I”加上 Pascal 形式的表示方式。

例如：IQuery, IDataAccess,IReportBuilder

* 变量名和参数名使用 Camel 表示方式。

userName, objectFactory, entrys, list

* 对于常量名，使用大写字母，并使用下划线做间隔。

如：MAX\_TIMES, DEFAULT\_NAME

程序中应该使用常量代替字符串，尤其是系统预定义的字符串。

如：if(service == Constants.COD) {…}

// 避免

if(service == “COD”) {…}

* 客户端页面初始化类放到包的form目录下，类名以FX结尾，继承FXClientSubUI父类。

页面详细布局，控件初始化，控件事件等处理逻辑放到form.FXUI目录下，类名以Controller结尾，继承FXClientSubUIController父类。

查询页面继承FXBaseQueryController父类。

* Controller类中页面控件属性名称以 控件描述\_类型缩写 规则命名，

如：@FXML

private DatePicker beginDate\_dp; // 开始时间

## 编程实践

* 只导入明确需要的类(快捷键 ctrl+shift + o)

import java.util.List;

// 避免: import java.util.\*

* 一个变量要代表独立的意思，不要在其生命周期赋予它不同的概念。

int tempValue;

tempValue = maxValue;

…

…

tempValue = minValue;

…

tempValue = anotherValue;

tempValue 在生命周期内表示了各种各样的意图，增加理解代码的难度。 应该为每个独立概念定义单独的变量

int tempMaxValue;

int tempMinValue;

int tempAnotherValue;

* 循环变量应靠近循环体初始化

isDone = false

while(!isDone) { … }

//避免

isDone = false;

…

…

while(!isDone) { … }

* 避免长的布尔表达式，应换成多个更容易理解的表达式。

bool isFinished = (elementNo < 0) || (elementNo > maxElement);

bool isRepeatedEntry = elementNo == lastElement;

if (isFinished || isRepeatedEntry) { … }

// 避免

if ((elementNo < 0) || (elementNo > maxElement)|| elementNo == lastElement) { … }

* 类和方法的粒度保持适中，保持类的规模尽量短小，职责单一。小类有很多好处，易于设计，易于测试，易于理解。同样方法也要尽量的小，每个方法尽量不要超出 50行。
* 使用lambda表达式简化代码(java 8)

// 推荐

Platform.runLater(() -> {

FXUtil.showAlertUseControls("导出成功!");

hideProgress();

});

// 避免

Platform.runLater(new Runnable()

{

public void run()

{

FXUtil.showAlertUseControls("导出成功!");

hideProgress();

}

});

* 使用stream处理集合类(java 8)

// 推荐

List<WeightNation> list = getList();

List<WeightNation> newList = list.stream().map(w -> getLocalWeight(w))

.collect(Collectors.toList());

// 避免

List<WeightNation> list = getList();

List<WeightNation> newList = new ArrayList<WeightNation>();

for(WeightNation w : list) {

newList.add(getLocalWeight(w))  
}

* 尽量少用if等判断语句，减少逻辑分支

// 推荐

CheckBox allCheckBox = new CheckBox("全选");

BooleanProperty isSelect = new SimpleBooleanProperty();

isSelect.set(allCheckBox.isSelected());

// 避免

CheckBox allCheckBox = new CheckBox("全选");

BooleanProperty isSelect = new SimpleBooleanProperty();

If(allCheckBox.isSelected()) {

isSelect.set(true);  
} else {

isSelect.set(false);

}

## 互相check代码

* Check代码的范围：常见的规范问题、逻辑/业务实现问题、冗余代码优化问题。
* Check代码的目标：每千行代码至少应该找出10个以上需优化的问题。

## 代码重构

* 长方法
* 巨大类
* 冗余啰嗦的逻辑
* 不合理的实现方案