



开发规范

| | | | |
|------|--|-----|-----------|
| 版 本 | V1.0 | | |
| 记录编号 | | | |
| 秘密等级 | <input type="checkbox"/> A 绝密 <input type="checkbox"/> B 机密 <input checked="" type="checkbox"/> C 内部公开 | | |
| 编 制 | 陈彦吉 | 日 期 | 2009-11-6 |
| 审 核 | | 日 期 | |

北京北控伟仕软件工程有限公司

文件修订记录

| 版本号 | 修订状态 | 修订说明 | 修订日期 | 修订人 |
|-----|------|---------------|------------|-----|
| 1.0 | 新增 | | 2009-11-6 | 陈彦吉 |
| 1.1 | 修改 | 整合四险项目所有开发类规范 | 2018-09-28 | 李文斌 |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

注：修订状态：新增、修改、删除。
修订说明：注明修订章节、修订内容。

目 录

1 目的

为了避免因为模块不符合规范、和框架调整造成模块错误。约束开发人员在项目开发过程中遵守一定的开发规范，使模块开发水平能够达到预先设计标准，也为日后升级维护工作提供良好基础。

2 适用范围

- ◆ 北京市社会保险信息管理系统。
- ◆ 北京市社会保险财务管理系统。
- ◆ 北京市社会保险经办工作内控监督系统。
- ◆ 北京市社会保险稽核管理子系统。
- ◆ 北京市社会保险基金票据管理系统。
- ◆ 北京市社会保险问题单管理子系统。

3 适用职位

项目经理、研发经理及上述系统设计人员、开发人员、测试人员。

4 Web 开发规范

4.1 界面规范

4.1.1 分辨率

本系统支持的最佳分辨率为 1024×768

4.1.2 浏览器

本系统支持的浏览器为 Internet Explorer（IE）6

4.1.3 字体

1. 字号原则上使用 14 号；
2. 为保证能在一个界面中操作而不出现滚动条，可以使用 12 号字；
3. 字体使用粗体；
4. 按钮的文字不使用粗体；
5. 字体颜色原则上使用黑色；
6. 必输项字体颜色使用蓝色。

4.1.4 菜单

弹出新窗口时自动隐藏左侧菜单区域；鼠标单击除菜单区以外的任何地方，自动隐藏左侧菜单区域；关闭窗口保持上一窗口操作状态；当关闭最后一个窗口时，自动弹出左侧菜单区域；

4.1.5 窗口

1. 窗体内容尽量保证在一个最大化的不带滚动条的页面中完全显示；
2. 不允许出现窗体的横向滚动条；
3. 如果窗体内容过多，可以在一个最大化的带纵向滚动条的页面显示；
4. 如果确实需要出现滚动条，需要组长审核；
5. 如果窗体内容过少，应该不最大化，窗体大小根据界面页面确定，推荐的宽度和高度约比为 3:2；
6. 窗口位置居中；
7. 窗口大小不允许手工调整；
8. 这时窗体的高度和宽度需要在数据库的 DM_XTQX 表中配置；
9. 每个窗口都必须有标题；
10. 窗体的标题一般由主程序自动产生，对于主程序不能自动产生标题的窗口，需自行设置；
11. 窗体中的弹出窗口必须设置标题。

4.1.6 页面布局

Index.css

1. 页面布局必须整齐、美观；页面上的各项之间排列要匀称；页面中不允许出现任何颜色值及常用样式设置，常用样式尽可能的使用样式表进行规范处理、或者提升为页面样式。避免出现多余的 `style` 设置，新增加样式表要即使加以注释。
2. 页面布局应参照老四险程序的界面，原则上与老四险界面相同；
3. 页面内容距窗体边缘保持相同的间距（间距原则为 15 个像素） 现需要在 BODY 内第一层增加样式 `sxDivStyle` ；（居中、上边 10PX 下边 5PX）例如：
4. 页面一般可能包括"查询条件区"、"查询结果区"、"操作按钮区"，每个区域之间应保持相同的间距（间距原则上为 15 个像素） 现统一引用样式 `trEmpty` （5 像素）
5. 查询区域中第一行设置的"查询图标"图片，更改为无图片默认背景色，避免取消图片后造成布局混乱。#E3EFFF
6. 取消每个区域的边框阴影效果（3D 效果）修改为 1 像素边框，即：
`border-bottom:#B5B5B5 1px solid;`
`border-top:#B5B5B5 1px solid;`
`border-left:#B5B5B5 1px solid;`
`border-right:#B5B5B5 1px solid;`
无边框背景区域应使用 `sxBcolor` 如：
`<table class="sxBcolor" 无边框主体色`
有边框背景区域使用 `sxBcolor-line` 如：
`<table class="sxBcolor-line" 有边框主体色`

7. 取消模块提示说明及区域说明文字。

4.1.7 表单

1. 必须设置属性 `<table class="table_black" cellspacing="1">` 可适当增加其他属性，但保证页面完整性。
2. 多行的表单底色交替变化，奇数为浅色（样式为 `black_td1 #ffffff`）、偶数为深色（样式为 `black_td2 #EEF5FF`） 例如：在`<tr>`上整体调整

`<tr calss="black_td1">` 浅色（自动调整内部 INPUT）

`<tr calss="black_td2">` 深色（自动调整内部 INPUT）

`<tr calss="black_td1_no">` 浅色

`<tr calss="black_td2_no">` 深色

也可以使用`<td>`进行调整

要求：复杂表单标签区域应使用交替色调。简单表单表头区域应使用与 GRID 同一颜色样式为：`<tr class="black_hd">` # E3EFFF

也可以使用原由交替样式深： `fuscoustr` 浅：`fleettr`（不完全）

3. 多行的表单每行保持相同的高度（行高原则上为 30）；宽度自定义（尽量做到不换行）
4. 表单项的说明文字默认居右齐，与控件之间保持一空格，（ ）
5. 必填项的说明文字采用蓝色字体（不加星号）
6. 表单项的说明文字至少留出 4 个字的宽度，如果无法完全显示可以折行，说明文字的排放应为：

单行 3 个字

单行 4 个字

单行 5 个字

第一行 3 个字、第二行 3 个字

第一行 3 个字、第二行 4 个字

第一行 4 个字、第二行 4 个字

第一行 4 个字、第二行 5 个字

第一行 5 个字、第二行 5 个字

.....

7. 表单项可以使用"回车"导航；可以使用 Tab 键进行正序或逆序的导航

8. 数字右对齐，其他项左对齐，日期项居中显示，并保持规定大小，基本

大小为：60px 90px 120px 150px.....

金额和人数的数值长度要预留出最大值，保证能够显示完整

日期项格式为 YYYY.MM.DD 或 YYYY.MM dateStyle 统一为 90PX

各种表单项的样式：统一修改控件边框颜色等

输入框：无需手动添加样式

复选框：checkboxstyle

单选框：checkboxstyle

只读区域：readonlyinputstyle

带 LOV 的文本项：lovinputstyle

仿 DIV 的文本项：divInput 或 divNameInput

下拉框：转换为 Ext 的下拉框或使用 Ext 原生 Combobox

9. 表格 grid

统一设置个人不需要进行处理。

表头高度与表格内容行高度一致

表格颜色交替变化

如果表格只作为查询用，不能通过键盘新增记录

表头和表格内容字体原则上为 14 号字

数字列右对齐，其他列左对齐

金额和人数的数值长度要预留出最大值，保证能够显示完整（特别是合计值项目）

日期项格式为 YYYY.MM.DD 或 YYYY.MM

除了特殊情况外，查询均采用分页形式，每页显示 20 条记录

备注：修改页面过程中不建议使用颜色替换，应使用样式进行处理。为维护方便，避免布局极复杂的层次嵌套。以简洁并能达到目的为原则。

4.1.8 按钮

1. 操作按钮通常在屏幕的下方靠右；
2. 按钮并排放置；
3. 按钮之间保持统一间距（间距原则上为 15 个像素）；
4. 按钮样式的样式统一为：` * * `；
5. 按钮需要加快捷键，样式不变统一设置；
6. 特别注明：对话框按钮取消快捷键。

4.1.9 清空按钮功能

清空按钮功能，统一采用 reload 方法，同时设置控件的初始值。修改部分系统已经实现的用程序清空控件内容的功能。

4.2 JSP 规范

1. 统一引入公共页面`<%@ include file="/pages/commons/meta.jsp"%>`，公共文件包括：EXT 支持、公共 JS、打印、消息提示、安全、公共样式。
2. 允许复制、改写`/pages/commons/meta.jsp`
3. 使用页面全局标签控制`<div class="sxDivStyle">`，作为 BODY 内第一层布局控制。
4. JSP 中不允许出现`<%=request.getContextPath()%>`可使用预先声明的`<`

`%=contextPath %>`，JS 中使用预定义 `var contextPath;`

5. 页面书写可省略<head>部分以减少代码量。
6. 控件必须有 ID 属性，并通过 ID 获取控件对象。
7. 避免重复引用、多余引用 JS 或其他文件。
8. 页面布局使用<table>控制，操作标签使用<div>。
9. 使用预定义 CSS 样式，样式参考应用菜单目录下：“样式规范”。
10. 文件命名与模块名称一致，文件的目录结构符合模块层次结构，文件名避免出现使用数字区分（如：abcd1.jsp）。必须保证当前目录下必须保证存在同名的 JS 文件（即使是空的）。
11. 控件命名尽可能遵守命名规则：用文件名作为前缀或缩写做为前缀。
12. 使用 C 标签方式`${}`替代`<%=request.getAttribute("")%>`，可在 JSP 中引入：
`<%@ include file="/pages/commons/taglibs.jsp"%>`
13. 显示空格使用` `；回车使用`
`。
14. 避免标签或控件未结束造成的页面警告。
15. 不允许引入 JAVA 包、也不允许在 JSP 页面中包含 JAVA 代码。

4.3 CSS 规范

1. 字体：默认字体为宋体 14#粗体，（统一布局控制，无需定义）。如果界面内容非常多，那么可以将字体设置为宋体 12#字,但需要获得开发经理的许可。
2. 工作窗体默认宽度为 800px，（无滚动条）；如果窗体内容过多，则设定为 1020PX 或带滚动条。窗体上下左右边界距离最近的控件距离为 10 个像素，可由`<div class="sxDivStyle">`整体控制。
3. 大区域之间间隔为 10 个像素，可使用样式 `trEmpty`。
4. 表格高宽值以实际像素单位定义。
5. 表 单 无 边 框 ， table 间 距 为 零 。 `<table border="0" cellpadding="0"`

cellspacing="0">

6. LOV 的文本框的样式表为: lovinputstyle
7. 只读区域的样式表为: readingonlyinputstyle
8. 一般性文本框不需引用样式表
9. 按钮样式为: 如果标题文字为 2 个那么中间插入两个 。
10. 表单内容:
11. 标题项右对齐; 必填区分方法用星号表示, 星号放文本左边, 文字右对齐。表格颜色交替变化。奇数为浅色 (样式为 black_td1), 偶数为深色 (样式为 black_td2) 输入框为 inputtd; 文字录入或数据显示区域采用 inputtd; 表格行高 30; 宽度自定义 (尽量做到不换行); 下拉列表按钮右对齐。按钮默认右对齐, 复选框为 heckboxstyle 单选框为 checkboxstyle 下拉框用 ext 转换, 日期输入提示: title='yyyy.mm.dd'或 title='yyyy.mm'
12. 查询列表内容:
 - a) 列表形式的表单, 其表头对齐方式与表格内容对齐方式一致, 数字右对齐, 其他左对齐。金额和人数的数值长度要预留出最大值, 保证能够显示齐全;
 - b) 表头高度与下面内容行高度一致。

4.4 JS 规范

4.4.1 代码组织与风格

4.4.1.1 缩进

使用 Tab 缩进, 而不是空格键。

4.4.1.2 空行

适当的增加空行, 来增加代码的可读性。

在下列情况下应该有两行空行:

- 同一文件的不同部分之间；
- 在下列情况之间应该有一行空行：
- 方法之间；
 - 局部变量和它后边的语句之间；
 - 方法内的功能逻辑部分之间。

4.4.1.3代码块长度

- 每个代码块尽量控制在 1 个屏幕之内，方便浏览。一个方法的大小应尽量控制在 60 行（约为 1 页打印纸）之内。
- 源代码文件的大小应尽量控制以 800 行为标准。

4.4.1.4{ }

程序的分界符开括号“{”应放置在所有者所用行的最后，其前面留一个空格；闭括号“}”应独占一行并且与其所有者位于同一列，需在“}”后加“;”加以结束。

```
SomeStatements;  
  
    for(var i = 0; i < n; i++) {  
        DoSomething();  
    };  
  
    .....
```

if,for,while 语句只有单句时，如果该句可能引起阅读混淆，需要用“{”和“}”括起来，并且缩进要听译。

4.4.1.5行宽

代码行最大长度宜控制在 80 个字符以内。代码行不要过长，否则眼睛看不过来，也不便于打印。

长表达式要在低优先级操作符处拆分成新行，操作符放在新行之首（以便突出操作符）。拆分出的新行要进行适当的缩进，使排版整齐，语句可读。

例如：

```
if((very_longer_variable1 >= very_longer_variable12)
    && (very_longer_variable3 <= very_longer_variable14)
    && (very_longer_variable5 <= very_longer_variable16)) {
    dosomething();
}
```

4.4.1.6 空格

- A. 方法名之后不要留空格，紧跟左括号“（”，以与关键字区别。
- B. “（”向后紧跟，“）”、“，”、“；”向前紧跟，紧跟处不留空格。
- C. “，”之后要留空格，如Function XXX(x, y, z)；如果“；”不是一行的结束符号，其后要留空格，如for (initialization; condition; update)。
- D. 赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=”、“>=”、“<=”、“+”、“*”、“%”、“&&”、“||”、“<<”，“^”等二元操作符的前后应当加空格。
- E. 一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格。
- F. 诸如“[]”、“.”、“->”这类操作符前后不加空格。

4.4.1.7 注释的基本约定

注释将增加代码的清晰度。注释需简洁、明了。注释不应该包括其他的特殊字符。序言性注释一定要有，在JS文件的首部，JavaScript函数的首部都要描述功能、主要参数、特别问题，以提供总概性提示。重要代码处也应该有相应的注释，提示阅读者。可先写注释，后写代码。

4.4.1.8 注释类型

4.4.1.8.1 Js 文件注释

Js 文件注释采用 `/** */`，在 Js 文件的第一行处要求有必要的注释信息，包括：工程名；作者；创建日期；类功能描述（如功能、主要算法、内部各部分之间的关系、该类与其类的关系等，必要时还要有一些如特别的软硬件要求等说明）。

注释模版：

```
/**
 * Project: <项目工程名>
 * Comments: <对此类的描述，可以引用详细设计中的描述>
 * Author: <作者中文名或拼音缩写，禁用 Administrator 及拼音首字母简写>
 * Create Date: <创建日期，格式:YYYY-MM-DD>
 */
```

4.4.1.8.2 变量注释

变量注释采用 `/***/`，包含全局和局部变量，在定义变量的上一行出要求有注释，包括：变量的中文解释。

4.4.1.8.3 方法注释

方法注释采用 `/***/`，在定义方法或者过程的上一行处要求有注释信息，包括：功能描述、参数说明、返回值说明、异常等。

注释模版：

```
/**
 * Comments: <方法的功能说明>
 * @param: <按照参数定义顺序，@param 后面空格后跟着参数的变量名字，
```

```
*           空格后跟着对该参数的描述。> ;
* @return:   <方法返回类型，无法返回值的方法可省略>;
* @author:   <方法作者，如果与 js 文件作者一致，可省略>;
* Create Date: <方法创建时间，如果与 js 文件同一天创建可省略> ;
*/
```

4.4.1.8.4 方法内部注释

方法内部采用 `/***/`,主要的代码功能描述等，特别是复杂的逻辑处理部分，要尽可能的给出详细的注释。

注：为避免 JavaScript 代码压缩造成的注释错误，不允许使用 “`/**`”注释。

4.4.1.8.5 推荐的注释内容

- 对于调用复杂的功能处理尽量提供代码示例。
- 对于已知的 Bug 需要声明。

4.4.1.8.6 特殊的注释内容

- 代码质量不高但能正常运行，或者还没有实现的代码用`/*TODO:*/`声明

4.4.1.9 代码格式化

- 使用 myeclipse 默认或 spket 默认格式化均可，使用 myEclipse 编辑器打开后按 `ctrl+shift+f` 进行格式化。

4.4.2 命名

4.4.2.1 命名的基本约定

本章定义了标识符（包括文件名、变量、参数、方法等的名称）的命名通用性原则。

4.4.2.1.1 原则一：充分表意

标识符应当直观且可以拼读，可望文知意，不必进行“解码”。标识符最好采用英文单词或其组合，便于记忆和阅读。切忌使用汉语拼音来命名。程序中的英文单词一般不会太复杂，用词应当准确。例如不要把 CurrentValue 写成 Now Value。标识符的长度应当符合“min-length && max-information”原则。在表示出必要的信息的前提下，标识的命名应该尽量的简短。例如，例如标识最大值的变量名命名为成 maxVal，而不推荐命名为 maxValueUntilOverflow。单字符的名字也是可用的，常见的如 i, j, k, m, n, x, y, z 等，它们通常可用作函数内的局部变量，如循环计数器等。

用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。

例如：

```
var minValue, maxValue;  
  
Function SetValue(...);  
  
Function GetValue(...);
```

单词的缩写应谨慎使用。在使用缩写的同时，应该保留一个标准缩写的列表，并且在使用时保持一致。尽量避免名字中出现数字编号，如 Value1、Value2 等，除非逻辑上的确需要编号。为了防止某一软件库中的一些标识符和其它软件库中的冲突，可以为各种标识符加上能反映软件性质的前缀。

4.4.2.1.2 原则二：避免混淆

程序中不要出现仅靠大小写区分的相似的标识符。

例如：

```
var x, X; // 变量 x 与 X 容易混淆  
  
Function foo(x); // 函数 foo 与 F00 容易混淆  
  
Function F00(x);
```

程序中不要出现标识符完全相同的局部变量和全局变量，尽管两者的作用域

不同而不会发生语法错误，但会使人误解，JavaScript 不存在重载或重写方法，默认调用同名第一个加载方法。

4.4.2.1.3 原则三：使用正确的词性

变量的名字应当使用“名词”或者“形容词+名词”。

例如：

```
var value;  
var oldValue;  
var newValue;
```

全局函数的名字应当使用“动词”或者“动词+名词”（动宾词组）。类的成员函数应当只使用“动词”，被省略掉的名词就是对象本身。

例如：

全局函数

```
drawBox=function() {  
    function draw() {};  
};
```

类的成员函数

```
drawBox.draw();
```

4.4.2.2 命名的一般规则

命名规则是针对文件名、包名、类名、接口名、方法名、变量名等名称而制定的。下面是适用于所有这些名称的规则。

- 只能采用可以用 ASCII 码表示的大写拉丁字母、小写拉丁字母、数字、下划线组成 3 个字符以上的命名规则。但是用 i、j、k 等作为局部循环变量是允许的——然而，使用意义明确的循环变量名来代替 i、j、k 等无疑是更好的做法。
- 第一个字符禁止使用下划线（“_”）。

- 避免使用下划线(静态变量除外)。
- 在任何的位置上禁止使用“\$”。
- 除了包名称之外，不允许使用国家代码和域名。
- 禁止使用仅由通用单词组成的不能表达被命名对象特征的名称。如，number、theInteger_、list、float_array 等。
- 禁止使用脚本、Java、C、C++ 等各种语言规则中的关键词。如，if、operator、define 等。
- 命名尽量小于 15 字符。

4.4.2.3 标示符的命名约定

4.4.2.3.1 文件名

- 文件名都应加上适当的扩展名以表示文件的类型。
- JavaScript 的文件名应遵循同目录 JSP 命名规范，与 JSP 命名保持一致。

4.4.2.3.2 目录名

- 全部小写。
- 清晰简洁，符合模块功能级目录划分。

4.4.2.3.3 方法

4.4.2.3.4

- 第一个单词一般是动词或者动词词组。
- 第一个单词小写，随后的各个单词的第一个字母大写其余字母小写的字符串。
- JavaScript 中不存在方法重写及重载，避免方法名重复造成的调用混淆。
- 为了方法灵活化，可使用可选参数（可空参数）。
- 如果方法返回一个成员变量的值，方法名一般为 get+成员变量名，如若返回的值是 boolean 变量，一般以 is 作为前缀。如果方法修改一个成员

变量的值，方法名一般为：set + 成员变量名。

如：getName(); setName(); isFirst();

4.4.2.3.5 属性名

- 属性的名字应该使用描述性的名词或名词词组。
- 非 final 的属性的名称必须为第一个单词小写，随后的各个单词的第一个字母大写其余字母小写。
- final 的属性的名称以全大写字母表示，各单词之间用下划线（“_”）来连接。

4.4.2.3.6 变量

4.4.2.3.7

- 局部变量和临时参数的名称由全部字母小写的一个或多个单词构成。
- 使用简短而富有含义的单词为局部变量及临时参数命名。
- 局部变量及输入参数不要与类成员变量同名(get/set 方法与构造函数除外)

4.4.2.3.8 常量

4.4.2.3.9

- JavaScript 中并没有常量标示，但可以声明一些变量人为控制为常量。
- 所有常量名均全部大写，单词间以 ‘_’ 隔开。
- 常数名的组成要素中的字母全部大写。常数名的组成要素可以是单词、多个单词的第一个字母的罗列、单词的缩略形式等。
- 常数名是记述性的，无必要时不要省略。
- 多个常数组成一个群体时，必须使用适当的短前缀。

如：var MAX_NUM;

4.4.3 声明

- 每个变量需要有一个声明。
- 局部变量必须初始化。
- 除了 for 循环外，声明应该放在块的最开始部分。for 循环中的变量声明可以放在 for 语句中。如：`for(var i = 0; i < 10; i++)`。
- 避免块内部的变量与它外部的变量名相同。
- 避免不同函数之间的命名冲突。

4.4.4 架构编程规范

4.4.4.1 基本规范

- 构造页面基本使用 EXT 常用组建。
- 每个 JSP 目录都应该有同名的 JS 文件(即使 JS 未空也要存在)。
- 页面加载方法使用 EXT 提供的 `Ext.onReady(function() { })`；
不允许使用其他方式加载、初始化代码。
- 避免 EXT 对象及监听的重复创建。
- 避免方法名、公共变量的命名与其他 JS 文件中命名冲突，公共变量使用标示作为命名前缀，如：`var dwgl_dwid`；
- 提倡使用 `$("#xxx")` 获取控件对象。
- 对没有进行传递的参数，参数使用前加以判断，`if(object != "undefined")`。
- 在模块操作整个工程中，不允许出现 JS 警告或错误提示。
- 有效性验证使用公共验证 JS：`/scripts/validate.js`
- 有效性验证代码使用正则表达式进行校验。
- 使用 EXT 封装消息功能（使用 `/scripts/vstMessage.js`），
不允许使用 `alert()`。
- 在有数据交互的方法首部增加安全验证和代码：`if(!validateKey())`

```
{return false; }
```

- 使用已有的公共函数或公共变量。
- 不允许复制与篡改公共函数或公共变量。
- 使用公共函数避免事件的重复提交。
- 针对不用请求提交方式，在嵌套提交部分，需设置为同步请求（默认 Ajax 为异步请求），DWR、EXT、BUFFALO、普通 AJAX 的同步异步设置请参照《01.解决方案\技术组-1416\ext、buffalo、dwr 同步异步.txt》
- WEBLOGIC 下 EXT 请求编码需特殊处理，请参照《01.解决方案\技术组-1416\ext 在 weblogic 下乱码问题.txt》
- 针对部分可能为空的对象、变量需要进行判断：if(typeof boo=="undefined")
-
- 部分 EXT 附加功能，请参组内代码、技术组咨询。
- 减少代码复杂程度，提升代码的可读性。

4.4.4.2 字符集与编码

- JavaScript 源代码、jsp 文件、以及相关配置文件的编码，均采用 utf-8
- 任何 JavaScript 代码不需要对编码进行处理。

4.4.4.3 异常处理

- 适当的使用 try{ }catch(e){} 进行异常捕获。多数用在对某空间的设置操作。捕获后不需要对异常进行处理。

AJAX 请求异常已使用公共化提示代码，不需要认为干涉。（提示消息为后台抛出异常信息。针对后台错误消息不明确的，默认 JavaScript 一律使用“由于网络不稳定或中断，请重新登录”）

5 JAVA 开发规范

5.1 代码组织与风格

5.1.1 缩进

使用 Tab 缩进，而不是空格键。

5.1.2 空行

适当的增加空行，来增加代码的可读性。

在下列情况下应该有两行空行：

- 同一文件的不同部分之间；
- 在类，接口以及彼此之间。

在下列情况之间应该有一行空行：

- 方法之间；
- 局部变量和它后边的语句之间；
- 方法内的功能逻辑部分之间。

5.1.3 代码块长度

- 每个代码块尽量控制在 1 个屏幕之内，方便浏览。一个方法的大小应尽量控制在 60 行（约为 1 页打印纸）之内。
- 源代码文件的大小以 800 行为标准，最大 1500 行到 3000 行左右。

5.1.4 { }

程序的分界符开括号“{”应放置在所有者所用行的最后，其前面留一个空格；闭括号“}”应独占一行并且与其所有者位于同一列。

```
SomeStatements;
```

```
    for(int i = 0; i < n; i++) {  
        DoSomething();
```

```
}  
.....
```

if,for,while 语句只有单句时，如果该句可能引起阅读混淆，需要用"{"和"}"括起来，否则可以省略。

5.1.5 行宽

代码行最大长度宜控制在 80 个字符以内。代码行不要过长，否则眼睛看不过来，也不便于打印。

长表达式要在低优先级操作符处拆分成新行，操作符放在新行之首（以便突出操作符）。拆分出的新行要进行适当的缩进，使排版整齐，语句可读。

例如：

```
if((very_longer_variable1 >= very_longer_variable12)  
    && (very_longer_variable3 <= very_longer_variable14)  
    && (very_longer_variable5 <= very_longer_variable16)) {  
    dosomething();  
}
```

5.1.6 空格

A. 关键字之后要留空格。诸如 const、virtual、inline、case 等关键字之后至少要留一个空格，否则无法辨析关键字。诸如 if、for、while 等关键字之后留一个空格再跟左括号“（”，以突出关键字。

B. 方法名之后不要留空格，紧跟左括号“（”，以与关键字区别。

C. “（”向后紧跟，“）”、“，”、“；”向前紧跟，紧跟处不留空格。

D. “，”之后要留空格，如 Function(x, y, z)；如果“；”不是一行的结束符号，其后要留空格，如 for (initialization; condition; update)。

E. 赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如

“=”、“+=” “>=”、“<=”、“+”、“*”、“%”、“&&”、“||”、“<

<”，“^”等二元操作符的前后应当加空格。

F. 一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格。

G. 诸如“[]”、“.”、“->”这类操作符前后不加空格。

5.1.7 注释的基本约定

注释将增加代码的清晰度。注释需简洁、明了。注释不应该包括其他的特殊字符。序言性注释一定要有，在 JSP 文件的首部，Java 类和方法的首部都要描述功能、作者、主要参数、特别问题，以提供总概性提示。重要代码处也应该有相应的注释，提示阅读者。可先写注释，后写代码。

5.1.8 注释类型

5.1.8.1 类注释

类注释采用 `/** */`，在定义类的上一行处要求有必要的注释信息，包括：工程名；作者；创建日期；类功能描述（如功能、主要算法、内部各部分之间的关系、该类与其类的关系等，必要时还要有一些如特别的软硬件要求等说明）。

注释模版：

```
/**
 * Project: <项目工程名>
 * Comments: <对此类的描述，可以引用详细设计中的描述>
 * Author: <作者中文名或拼音缩写，禁用 Administrator 及拼音首字母简写>
 *
 * Create Date: <创建日期，格式:YYYY-MM-DD>
 */
```


5.1.8.2 变量注释

变量注释采用 `/***/`，包含全局和局部变量，在定义变量的上一行出要求有注释，包括：变量的中文解释。

5.1.8.3 方法注释

方法注释采用 `/***/`，在定义方法或者过程的上一行处要求有注释信息，包括：功能描述、参数说明、返回值说明、异常等。

注释模版：

```
/**
 * Comments: <方法的功能说明>
 * @param:    <按照参数定义顺序，@param 后面空格后跟着参数的变量名字
 *            (不是类型)，空格后跟着对该参数的描述。> ;
 * @return:    <方法返回类型，返回为空（void）的方法，可省略>;
 * @throws:    <方法抛出异常类型> ;
 * @author:    <方法作者，如果与类作者一致，可省略>;
 * Create Date: <方法创建时间，如果与类同一天创建可省略> ;
 */
```

5.1.8.4 方法内部注释

方法内部采用 `/***/`或`//.....`,主要的代码功能描述等，特别是复杂的逻辑处理部分，要尽可能的给出详细的注释。

5.1.8.5 推荐的注释内容

- 对于调用复杂的 API 尽量提供代码示例。
- 对于已知的 Bug 需要声明。
- 在本方法中抛出的 unchecked exception 尽量用 @throws 说明。

5.1.8.6特殊的注释内容

- 代码质量不高但能正常运行，或者还没有实现的代码用//TODO:声明
- 存在错误隐患的代码用//FIXME:声明
- 类和方法的注释模板可通过 Eclipse IDE 中的 windows-->preference-->Java-->Code Style-->Code Templates 进行相应设置。

5.2 命名

5.2.1 命名的基本约定

本章定义了标识符（包括文件名、变量、参数、方法等的名称）的命名通用性原则。

5.2.1.1原则一：充分表意

标识符应当直观且可以拼读，可望文知意，不必进行“解码”。标识符最好采用英文单词或其组合，便于记忆和阅读。程序中的英文单词一般不会太复杂，用词应当准确。例如不要把 CurrentValue 写成 NowValue。标识符的长度应当符合“min-length && max-information”原则。在表示出必要的信息的前提下，标识的命名应该尽可能的简短。例如，例如标识最大值的变量名命名为 maxVal，而不推荐命名为 maxValueUntilOverflow。单字符的名字也是可用的，常见的如 i, j, k, m, n, x, y, z 等，它们通常可用作方法内的局部变量，如循环计数器等。

用正确的反义词组命名具有互斥意义的变量或相反动作的方法等。

例如：

```
int minValue, maxValue;  
  
int SetValue(...);  
  
int GetValue(...);
```

单词的缩写应谨慎使用。在使用缩写的同时，应该保留一个标准缩写的列表，

并且在使用时保持一致。 尽量避免名字中出现数字编号，如 Value1、Value2 等，除非逻辑上的确需要编号。为了防止某一软件库中的一些标识符和其它软件库中的冲突，可以为各种标识符加上能反映软件性质的前缀。

5.2.1.2 原则二：避免混淆

程序中不要出现仅靠大小写区分的相似的标识符。

例如：

```
int x, X; // 变量 x 与 X 容易混淆  
void foo(int x); // 方法 foo 与 F00 容易混淆  
void F00(double x);
```

程序中不要出现标识符完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发生语法错误，但会使人误解。

5.2.1.3 原则三：使用正确的词性

变量的名字应当使用“名词”或者“形容词+名词”。

例如：

```
int value;  
int oldValue;  
int newValue;
```

全局方法的名字应当使用“动词”或者“动词+名词”（动宾词组）。类的成员方法应当只使用“动词”，被省略掉的名词就是对象本身。

例如：

```
drawBox(); // 全局方法  
Box.draw(); // 类的成员方法
```

5.2.2 命名的一般规则

命名规则是针对文件名、包名、类名、接口名、方法名、变量名等名称而制

定的。下面是适用于所有这些名称的规则。

- 只能采用可以用 ASCII 码表示的大写拉丁字母、小写拉丁字母、数字、下划线组成 3 个字符以上的 Java 语言名称。但是用 i、j、k 等作为局部循环变量是允许的——然而，使用意义明确的循环变量名来代替 i、j、k 等无疑是更好的做法。
- 第一个字符禁止使用下划线（“_”）。
- 避免使用下划线(静态变量除外)。
- 在任何的位置上禁止使用“\$”。
- 除了包名称之外，不允许使用国家代码和域名。
- 禁止使用仅由通用单词组成的不能表达被命名对象特征的名称。如，number、theInteger_、list、float_array 等。
- 禁止使用 Java、C、C++ 等各种语言规则中的关键词。如，if、operator、define 等。
- 命名尽量小于 15 字符。

5.2.3 标示符的命名约定

5.2.3.1 文件名

- 文件名都应加上适当的扩展名以表示文件的类型。
- 源代码的文件名必须与该文件内的公开类(Public Class)的类名相同，且扩展名应取“.java”。

5.2.3.2 包

5.2.3.3

- 全部小写。
- 为了保证包名是唯一的，要附加反转顺序的域名前缀。例如，cn.com.XXXX.XXXX。

5.2.3.4类名和接口

5.2.3.5

- 类的名字应该使用描述性的名词或名词词组。
- 类名必须为各单词的第一个字母大写其余字母小写的字符串。
- 类名不要长得过度。
- 尽可能避免使用与 JDK 标准类库相同的名字。例如，ClassLoader。
- 接口尽量采用"able", "ible", or "er", 如 Runnable 命名, 尽量不采用首字母为I或加上 IF 后缀的命名方式, 如 IBookDao,BookDaoIF。

5.2.3.6方法

5.2.3.7

- 第一个单词一般是动词或者动词词组。
- 第一个单词小写, 随后的各个单词的第一个字母大写其余字母小写的字符串。
- 如果方法返回一个成员变量的值, 方法名一般为 get+成员变量名, 如若返回的值是 boolean 变量, 一般以 is 作为前缀。如果方法修改一个成员变量的值, 方法名一般为: set + 成员变量名。

如: getName(); setName(); isFirst();

5.2.3.8属性名

- 属性的名字应该使用描述性的名词或名词词组。
- 非 final 的属性的名称必须为第一个单词小写, 随后的各个单词的第一个字母大写其余字母小写。
- final 的属性的名称以全大写字母表示, 各单词之间用下划线 (“_”) 来连接。

5.2.3.9 变量

5.2.3.10

- 局部变量和临时参数的名称由全部字母小写的一个或多个单词构成。
- 使用简短而富有含义的单词为局部变量及临时参数命名。
- 局部变量及输入参数不要与类成员变量同名(get/set 方法与构造方法除外)

5.2.3.11 常量

5.2.3.12

- 所有常量名均全部大写，单词间以 ‘_’ 隔开。
- 常数名的组成要素中的字母全部大写。常数名的组成要素可以是单词、多个单词的第一个字母的罗列、单词的缩略形式等。
- 常数名是记述性的，无必要时不要省略。
- 多个常数组成一个群体时，必须使用适当的短前缀。

如：int MAX_NUM;

5.3 声明

- 每行应该只有一个声明。
- 修饰符应该按照如下顺序排列：public, protected, private, abstract, static, final, synchronized。
- 类与接口的声明顺序(可用 Eclipse 的 source->sort members 功能自动排列)
 - 静态成员变量 / Static Fields
 - 静态初始化块 / Static Initializers
 - 成员变量 / Fields
 - 初始化块 / Initializers
 - 构造器 / Constructors

- 静态成员方法 / Static Methods
- 成员方法 / Methods
- 重载自 Object 的方法如 toString(), hashCode() 和 main 方法
- 类型(内部类) / Types(Inner Classes)

同等的类型，按 public, protected, private 的顺序排列。

- 局部变量必须初始化。
- 除了 for 循环外，声明应该放在块的最开始部分。for 循环中的变量声明可以放在 for 语句中。如：for(int i = 0; i < 10; i++) 。
- 避免块内部的变量与它外部的变量名相同。

5.4 编程规范

5.4.1 基本规范

- 正式代码中不能使用 System.out.println(), e.printStackTrace(), 必须使用 logger 打印信息。
- 变量，参数和返回值定义尽量基于接口而不是具体实现类，如 Map map = new HashMap();
- 用 double 而不是 float
- 对于精确数值运算时使用 BigDecimal 而不是 double
- 隐藏工具类的构造器，确保只有 static 方法和变量的类不能被构造实例。
- 在数组中的元素(如 String [1])，如果不再使用需要设为 NULL，直接用 Collections 类而不是数组。
- 尽量使用 protected 而不是 private，方便子类重载。

5.4.2 字符集

- Java 源代码、jsp 文件、以及相关配置文件的编码，均采用 utf-8

5.4.3 异常处理

- 重新抛出的异常必须保留原来的异常，即 `throw new NewException("message", e);` 而不能写成 `throw new NewException("message")`。
- 在所有异常被捕获且没有重新抛出的地方必须写日志。
- 如果属于正常异常的空异常处理块必须注释说明原因，否则不允许空的 `catch` 块。

5.4.4 JDK5.0 规范

- 重载方法必须使用 `@Override`，可避免父类方法改变时导致重载方法失效。

不需要关心的 warning 信息用 `@SuppressWarnings("unused")`, `@SuppressWarnings("unchecked")`, `@SuppressWarnings("serial")` 注释。

6 框架规范

6.1 Dwr 规范

1. Java 中不要重载需要在 JavaScript 端调用的方法，Dwr 不能确认具体调用哪个方法。
2. JavaScript 端回调函数名称应该是一个对象即函数对象而不是 Java 方法参数。

如： `*****Service.testDwr(par1,par2,callbackFunName);`
或者 `*****Service.testDwr (par1,par2, function(data){`
`.....`
`});`

而非 `*****Service.testDwr (par1,par2,“callbackFunName”);`

3. Java 类名和方法名不能以 Java 和 JavaScript 的关键字命名，JavaScript 的

关键字和 Java 是大多相同的,但是还有一些不同, 如: delete;

6.2 Struts 规范

4. 为 Action 文件配置 BEAN, 使用 name 属性定义 BEAN 名称。如: `<bean name="/cwgl/skcl/skcx" class="com.vstsoft.csi.web.cwgl.skcl.SkcxAction"/>`
5. Struts 配置文件统一使用 `name="mainForm" scope="request" parameter="method" validate="false"`, mainForm 为全局对象。
6. Action 文件以 ActionForward 作为返回值的方法只负责页面跳转, 不具备数据库操作功能(不允许在 Action 中使用 `manager.ibatisMethod`)。
7. 页面使用 Buffalo 异步技术进行数据操作, 具有数据库操作调用 Action 中非 ActionForward 返回值的方法, 不允许 Buffalo 调用 manager 层。
8. Action 中使用 Spring 自动注入获取 Manager 实例, 并声明变量为 private
9. 避免 Struts 配置文件属性 path 名称重复。

6.3 Spring 规范

1. 不允许擅自修改 spring 公共配置文件。
2. Bean 属性文件总体配置遵循: `<beans default-lazy-init="false" default-autowire="byName">`
3. 除 Action 以外配置 Bean 使用 ID 作为标识, 如: `<bean id="vDwRdManager" class="com.vstsoft.csi.service.dwgl.VDwRdManager" />`
4. Spring 配置文件中的 Buffalo 配置 name 命名不能重复, 如: `<bean name="buffaloConfigDwglBean" class="net.buffalo.service.BuffaloServiceConfigurer">`
5. Buffalo 配置中 key 命名不能重复, 如: `<entry key="DwKhTzService"><ref bean="/dwKhTz" /></entry>`
6. 避免循环注入造成的严重错误。

6.4 Ibatis 规范

1. Sql-map 的命名空间不能冲突，如：<sqlMap namespace="CW_GSFK" >
2. Sql-map 的类对象定义名称不能冲突，如：<typeAlias alias="CwGsfk" type="com.vstsoft.csi.model.cwgl.CwGsfk" />
3. resultMap 命名不能冲突，如：<resultMap id="CwGsfkResult" class="CwGsfk" >
4. sql 语句命名不能冲突，如：<insert id="com.vstsoft.csi.model.cwgl.CwGsfk.insert" 或者 <delete id="com.vstsoft.csi.model.cwgl.CwGsfk.delete" 或者 <update id="com.vstsoft.csi.model.cwgl.CwGsfk.update"
5. 确保高效的 SQL，避免查询出重复列，避免使用 ROWNUM_ 关键字。
6. SQL-MAP 参数不允许使用 SQL 拼接方式作为参数（防止 SQL 注入），参数中不允许出现数据库表名及字段名，一律使用 SQL 参数预定义形式。
7. 尽量通过 IBatisEntityManager<T>进行数据库操作，使用泛型来确定 SQL-MAP 名称。

6.5 日志开发规范

统一调用系统中公用的日志存储公用方法，考虑到新四险技术架构的特殊性，公用日志方法实现了有 HttpServletRequest 参数和无 HttpServletRequest 参数两个方法。

6.5.1 总体规范

- ◆ 所有 Action 类必须继承于 **RoutingAction** 类
- ◆ 所有 Manager 类必须继承于 **IBatisEntityManager** 类

6.5.2 日志公用方法调用规范

- ◆ 无论是有 `HttpServletRequest` 参数和无 `HttpServletRequest` 参数的公用日志方法都实现在 `Action` 的父类 **`RoutingAction`** 和 `Manager` 的父类 **`IBatisEntityManager`** 类中，在各自的请求处理类(****`Action.java`) 和业务处理类(****`Manager.java`)中日志公用方法调用方式如下：

- 1) 有 `HttpServletRequest` 参数调用

```
this.saveLog(String qxdm,String gnsn, String bz, Class classObj,  
HttpServletRequest request,java.util.Date date);
```

- 2) 无 `HttpServletRequest` 参数调用

```
this.saveLog(String qxdm,String gnsn, String bz, Class  
classObj,java.util.Date date);
```

- ◆ 确保功能代码执行成功后记录日志(调用 `this.saveLog(...)`) ;
- ◆ 只允许使用 `this.`调用日志方法，任何使用 `getBean(..)`直接获得日志类对象的，将视为非规范写法。

6.5.3 参数传入规范

- ◆ 有 `HttpServletRequest` 参数

- 1) `qxdm`:权限代码，要求以 `DM_XTQX` 表中对应的 `QXDM` 字段值。
- 2) `gnsn`:功能说明，手工记录本次操作的功能中文说明，如：系统登录、人员增加、单位信息录入等，多数以页面按钮名称为准。
- 3) `bz`:备注，手工记录本次操作的重要参数数据，以斜杠 (/) 分割每个重要参数，单个参数形式为：“参数中文名称:参数值” 如：

个人 id:grid/单位代码:dwdm/ 增减日期: zjrq 等等。

此参数为日志的重点记录对象，注意点如下：

- 3.a、 根据实际业务情况要求完整的记录下本次操作的重要参数数据，该字段长度要求不能超过 4000 字符。

3.b、 对于系统中已经用参数名记录的如：（ grid:grid/
dwdm:dwdm/ zjrq: zjrq）暂不修改，维持现状。

3.c、 对于系统中未添加日志的业务方法或该参数记录不全的无论是修改或新增日志都要求遵守该参数传入规范。

- 4) classObj:当前 Action 或 Manager 类对象。
- 5) request:当前 HttpServletRequest 请求对象。
- 6) date:当前应用服务器时间，要求在调用方法开始时定义并在调用日志方法时作为参数传入。
- 7) true/false:公用日志方法返回值,日志方法执行是否成功，一般不需要调用方法接收。

◆ 无 HttpServletRequest 参数使用

除无 request 参数外，其他参数传入规范同上。

6.5.4 实例

系统登录：

```
this.saveLog("xtld","系统登陆","用户 id:"+this.getUserid()+"/区县编  
码:"+this.getQxbm()+"/登录时间:"+new java.util.Date(),this.getClass(),request);
```

6.5.5 异常处理

如记录日志期间由于日志导致系统出现异常将由日志父类捕获，并不会影响到正常程序。

6.6 Report 开发规范

Report 为报表程序开发使用工具。

6.6.1 命名

Report 文件的命名建议取与调用该 report 的应用程序同样的名称。

6.6.2 特别要求优先

1. 如果用户有特别的要求，必须严格执行。
2. 某些打印，从页面布局、字体、行距，甚至每行的字数，都必须严格按照需求的表样执行。
3. 报表要尽量设计成打印模板，显示数据量的多少及排序通过调用程序控制，不要写在报表程序里；除非报表布局界面相关内容调整，否则无需修改报表程序。
4. 报表程序尽量不要写代码。

6.6.3 纸张类型

纸张类型选择的优先顺序为：A4 纵向、A4 横向、窄行打印纸、宽行打印纸。

6.6.4 字体

1. 标题：宋体（chinese_gb2312），14 号，加粗；
2. 其他：宋体（chinese_gb2312），10 号

6.6.5 数据模型的 SQL 查询语句

1. SQL 查询语句中列不允许用*号表示，要写出具体的列；
2. 公式尽量写在 SQL 语句里；
3. SQL 查询语句不写任何查询条件及排序规则，只有一个 P_WHERE 参数；
例如：SELECT DM,MC FROM DM_LX &P_WHERE;
4. 不要只为了编译需要去创建临时表，可以用传入 SQL 语句代替；
例如：SELECT DM,MC FROM (&P_SQL) &P_WHERE;

6.6.6 循环框

1. 如果有表样，要严格按照表样设置循环框每页的循环次数；
2. 没有表样的，要在一页上尽可能多的打印数据；

3. 循环框填充色为无;

6.6.7 参数列表

避免让用户见到参数列表，除在调试 REPORT 外，一律要将参数列表隐藏。

6.6.8 参数

1. 参数要注意参数类型及长度，长度要足够容纳传入数据;
2. 报表 SQL 语句查询条件参数名为: P_WHERE，长度不小于 2000;
3. 参数一定要有缺省值;

6.6.9 文本项（域）

1. 数字型文本项居右;
2. 其它类型文本项居左;
3. 数字型（金额）文本项格式掩码为 fm999999990.90;
4. 数字型（其他整型数字）文本项格式掩码无要求;
5. 日期型文本项格式掩码为 YYYY.MM.DD 或 YYYY.MM;
6. 在循环框中的文本项，要放置在循环框的底部
7. 文本项要求填充色为无;
8. 文本项要求线条颜色为无;
9. 文本项要求文本颜色为 black;

6.6.10 页面设置

1. 整体布局上，要选择左右居中，垂直方向稍向上;
2. 上下左右边距，应不少于 2cm;
3. 如下图，页面可以分为六个部分:

| | |
|---|----|
| 1 | 页眉 |
| 2 | 标题 |
| 3 | 表头 |

| | |
|---|----|
| 4 | 表体 |
| 5 | 表尾 |
| 6 | 页脚 |

4. 页眉：一般不添加内容；
5. 表头：显示单位名称、数量单位等，表头与标题之间的间距应在 5mm，表头与表体之间的间距应在 3~5mm；
6. 表尾：显示打印人、打印日期等，表尾与表体之间的间距应在 3~5mm；
7. 页脚：显示页码，格式为“第 页 共 页”，右对齐；

6.6.11 注意事项

1. 修改程序时，对原来的代码不要轻易删除，除非你确认该代码是错误的或者是不需要的。**应采用注释的方法**，并注明注释人的员工号、姓名、日期、原因；
2. 在界面和打印中，**注意金额项目的长度**，特别是合计值项目。
3. 修改以前的程序时，不需要对其中不符合规范的代码进行修改。但是，新增加的代码必须执行本规范；
4. 程序调试是编程工作的重要步骤，程序员要尊重测试人员的工作，不得把未调试通过的程序交付测试人员；
5. 框线尽量使用框架缺省框线，减少画线数量；

6.7 事务规范

1. Action 文件中以非 ActionForward 返回类型的方法都具有自动事务代理功能，无法为 ActionForward 返回类型的方法代理事务。
2. 自动代理事务方法命名规则遵守：查询功能方法命名以 find*或 get*，默认事务级别为只读；不符合查询命名规则的其他方法按照自动代理事务

默认处理，传播行为、隔离级别都是数据库默认级别。

3. 需要回滚的事务，需在 Action 方法中抛出 runtimeException 或子类异常（异步处理会自动提示异常信息），抛出 Exception 不回滚。

6.8 定时任务配置文件编写规范

基于 Spring 的定时任务配置文件编写规范。

6.8.1 文件命名规则和路径位置

基于 Spring 的定时任务配置文件的命名规则为：

applicationContext-quartzXXXX.xml

其中 xxxx 为功能略写，如：

applicationContext-quartzWssb.xml，指网上申报定时任务配置文件。

applicationContext-quartzXxfb.xml，指信息发布配置文件。

将定时任务配置文件的命名规则定义如上，意义在于不同的业务功能不能写入同一定时任务配置文件中。

这些配置文件的路径位置为：

/newsx/src/resources/spring/

6.8.2 配置内容编写规范

以信息发布定时任务为例（红色部分为规范内容）：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean name="quartzSchedulerXxfb"
    class="com.vstsoft.csi.core.quartz.VstSchedulerFactoryBean">
```

<!-- bean name 命名方式为 quartzSchedulerXXXX，其中，XXXX 为功能略写，它必须与文件名中的功能略写一致。Bean name 不能与其他定时任务配置文件中的 Bean name

重名。Bean 对应的 class 必须是

```
com.vstsoft.csi.core.quartz.VstSchedulerFactoryBean -->
```

```
<property name="triggers">
    <list>
        <ref bean="xxfbTrigger" />
    </list>
</property>
```

<!-- 以上 triggers 属性的定义方式 -->

```
<property name="configLocation"
    value="classpath:resources/quartz.properties" />
```

<!-- 以上为 configLocation 属性定义方式，必须存在，且不能修改 -->

```
<property name="ips">
    <list>
        <value>xxx.xxx.xxx.xxx</value>
    </list>
</property>
```

<!-- 以上为可触发定时任务的 IP 配置属性定义方式。Xxx.xxx.xxx.xxx 为 IP 地址，如果 IP 段不足 3 位不能补 0。 可以通过多个<value></value>标签对设置多个 IP 地址。-->

```
</bean>
```

```
<bean id="xxfbTrigger"
    class="org.springframework.scheduling.quartz.CronTriggerBean">
    <property name="jobDetail" ref="xxfbDetail" />
    <property name="cronExpression" value="0 0/1 0-23 * * ?" />
</bean>
```

<!-- 以上为 trigger 的定义方式，trigger ID 命名规则为：XXXXTrigger????，其中 xxxx 表示功能略写，必须与文件名的功能略写一致；????表示详细功能说明，整个系统不能存在同样 id 的 trigger。 -->

```
<bean id="xxfbDetail"
```

```

class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
    <property name="targetObject" ref="/desktopConfig" />
    <property name="targetMethod" value="getXxfb2" />
    <property name="concurrent" value="false" />
</bean>

<!-- 以上为 detail 的定义方式，detail ID 命名规则为：XXXXDetail????，其中
xxxx 表示功能略写，必须与文件名的功能略写一致；????表示详细功能说明，整个系统不能存在同样 id 的 detail。 -->

</beans>

```

6.9 系统编码及导出编码

项目编码一向做为系统设计级别难题，servlet2.4 还未实现跨应用服务器编码支持，个应用服务器之间编码存在很大差异，因此导致部署不同应用服务器上会出现不同乱码问题。为了实现开发环境 (tomcat) 和生产环境 (weblogic9.2) 编码统一，特此实现 WEB.XML 配置 FILTER 过滤编码，经过多次测试，基本实现跨应用服务器编码统一。

6.9.1 配置（工程编码：UTF-8）

6.9.1.1 个人编码规范

1. 由于一些模块存在 GET 请求，在处理 request.getParameter 后出现乱码问题，一般个人实现方式为 dwmc = new String(dwmc.getBytes("iso-8859-1"), "gb2312"); 此转码形式只适合一种应用服务器 tomcat，为保证生产环境下 weblogic 能正常编码，完全清理 get 请求因个人书写而改变编码设置代码。不允许出现针对 GRID 进行 SET 属性传递参数，如下

```

store.on('beforeload', function() {
    store.baseParams = {
        dmLx : dmLx,

```

```
dmMx : dmMx
};
});
```

如果使用此方法有汉字传递，将会出现乱码问题。

2. 导出文件中文名的转换，应用过滤无法拦截模拟请求，仍保留原有编码转换样式，不做任何修改。如

```
response.setContentType("application/vnd.ms-excel;application/msexcel;charset=gbk")

new String((userInfo.getUserName() + "__ 补缴明细录入表").getBytes("gbk"), "iso8859-1")
```

3. 跨操作系统绝对文件路径解决方案，UNIX 下识别"/"，WINDOW 识别“\”。PMS 配置文件中增加 system.name 指定 window/unix 操作系统名称，由 pathUtil.getPath() 获得当前造作系统路径写法，针对 window 下 getRealPath 无法获得最后路径符号而解决。pathUtil.conversionPath 根据配置转换路径符号。

原代码：

```
String filePath =
request.getSession().getServletContext().getRealPath(java.io.File.separator) + "pages"
+ java.io.File.separator + "jjsj" + java.io.File.separator +
"jjbj" + java.io.File.separator;
```

更改后代码：

```
String filePath =
UnixAndWindowPathUtil.conversionPath(request.getSession().getServletContext().getRealPath(File.separator)
+UnixAndWindowPathUtil.getPath()
+ "pages" + File.separator + "jjsj" +
File.separator + "jjbj" + File.separator);
```

6.9.1.2 系统配置 TOMCAT

POST：系统 web.xml 中使用 filter 进行过滤 POST 请求，并其自动对其编码转

换为 utf-8。实现代码为

```
request.setCharacterEncoding(encode);
```

GET：filter 过滤无法为 TOMCAT 进行 GET 编码转换，因此需要修改 TOMCAT/CONF/ server.xml 文件，在 项目 Connector 标签下增加 URIEncoding="GB2312"属性。

6.9.1.3 系统配置 WEBLOGIC9.2

POST 与 GET 统一处理方式，使用 FILTER 进行处理。判断 request.getContentType() 空的时候则为 GET 请求，处理如下：

```
request.setCharacterEncoding(encode_wlc);
response.setContentType("application/x-www-form-urlencoded; charset="+ encode_wlc);
response.setCharacterEncoding(encode_wlc);
encode_wlc: 为配置参数，此处：GB2312

POST 请求则正常转换成 utf-8 即可，
request.setCharacterEncoding(encode);
```

6.9.2 web.xml 配置

```
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-
class>com.vstsoft.csi.util.SetCharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value> //默认 POST 编码规则
    </init-param>
    <init-param>
        <param-name>encoding_wlc</param-name>
        <param-value>GB2312</param-value> //默认 WEBLOGIC 下 GET 编码规则
    </init-param>
</filter>
<filter-mapping>
```



```

    <filter-name>encodingFilter</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>

```

6.9.3 JAVA 文件

```

package com.vstsoft.csi.util;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class SetCharacterEncodingFilter implements Filter {

    protected String encoding = null;
    protected String encoding_wlc = null;
    protected FilterConfig filterConfig = null;
    protected boolean ignore = true;

    public void destroy() {
        this.encoding = null;
        this.encoding_wlc=null;
        this.filterConfig = null;
    }

    public void doFilter(ServletRequest request, ServletResponse
response,
        FilterChain chain) throws IOException, ServletException {
        if (ignore || (request.getCharacterEncoding() == null)) {
            String encod = selectEncoding(request);
            String encod_wlc = selectEncoding_wlc(request);

```

```

        if (request.getContentType() == null) {
            request.setCharacterEncoding(encod_wlc);
            response.setContentType("application/x-www-form-urlencoded;
charset="+ encod_wlc);
            response.setCharacterEncoding(encod_wlc);
        } else {
            request.setCharacterEncoding(encod);
        }
        encod=null;
        encod_wlc=null;
    }
    chain.doFilter(request, response);
}

public void init(FilterConfig filterConfig) throws ServletException
{
    this.filterConfig = filterConfig;
    this.encoding = filterConfig.getInitParameter("encoding");
    this.encoding_wlc =
filterConfig.getInitParameter("encoding_wlc");
    String value = filterConfig.getInitParameter("ignore");
    System.out.println("=====>编码过滤初始化，默认 POST 请求编码：
"+this.encoding+", WBL 下 GET 请求编码： "+this.encoding_wlc+"<=====");
    if (value == null)
        this.ignore = true;
    else if (value.equalsIgnoreCase("true"))
        this.ignore = true;
    else if (value.equalsIgnoreCase("yes"))
        this.ignore = true;
    else
        this.ignore = false;
}

protected String selectEncoding(ServletRequest request) {
    return (this.encoding);
}

protected String selectEncoding_wlc(ServletRequest request) {
    return (this.encoding_wlc);
}

```

```
}  
}
```

6.9.4 UNIX 环境编码设置（开发环境）

默认 UNIX 操作系统编码为 NULL，导致应用中 BUFFALO 参数乱码，更改 UNIX 操作系统编码格式为：在环境变量中加入 `export LANG=zh_CN.gb18030` 来解决部分乱码问题。

6.9.5 模糊查询

模糊查询分页与 BUFFALO 绑定 FORM 冲突中文乱码问题解决办法：

修改模糊查询分页编码格式：

```
Ext-base.js ext.lib.ajax.request  
//chenyanji  
if(K) {  
    if(K=="POST") {  
        if(J) {  
            if(J.indexOf("start=")>=0&&J.indexOf("&limit=")>=0) {  
                this.defaultPostHeader="application/x-www-form-urlencoded; charset=GB2312";  
            } else {  
                this.defaultPostHeader="application/x-www-form-urlencoded; charset=UTF-8";  
            }  
        }  
    }  
}
```

6.9.6 其他

注意：GRID 的 beforeload 事件中不能设置中文参数，次参数无法被编译。

6.10 其它规范

1. BEAN 的命名不能重复。
2. SQL-MAP 中各个类型定义的命名不能重复。

3. STRUTS 配置 path 命名不能重复。
4. BUFFALO 命名空间不能重复。
5. 使用注入方式获取 BEAN 实例，必须定义成 private
6. 所有 BEAN 文件不能创建全局变量，系统涉及（action、manager、dao）
7. 遵守 BEAN 的命名规则的同时（前两位必须为小字母）JAVA 文件也要遵守命名规则（首字母大写，第二位字母小写）。
8. 不随便定义公共文件及公共方法。
9. 复制文件后应对方法、公共变量名称进行修改，避免混淆。
10. 不允许定义方法或变量时候加数字标识。如：exportExcelByModel2(..)
exportExcelByMode22(..)
11. 遵守自动代理事务的命名规则。

模块中的逻辑处理代码尽可能编写在系统中的 service 层（即***Manager 类中）。

7 其他

7.1 版本控制

1. 采用 SVN 作为版本控制工具。
2. 进行开发工作之前要求先更新代码。
3. 禁止提交未通过编译或未编译的代码。
4. 文件提交时要求填写注释，注释内容清晰描述本次提交内容，变动信息等。

7.2 性能调优

目的是阐述在系统使用过程中对性能运行较慢的数据库语句的监控、收集、传递、解决的工作流程和注意事项，以规范调优工作，保障调优级质量和效率。

7.2.1 命名及规定

7.2.1.1 问题命名规定

问题内容：包含三部分，SQL 详细资料、原有语句执行计划、新的执行计划，该内容因为用监控工具产生，以 BMP 格式保存。

命名规则：日期-问题编号.问题内容

举例：在 2009-3-1 对问题 1 的监控

20090301-1.1.bmp (说明：SQL 详细内容)

20090301-1.2.bmp (说明：原有语句执行计划)

20090301-1.3.bmp (说明：新的执行计划)

注意：如需补充或增加新的文件，请按序号依次命名。

7.2.1.2 问题分析结果命名规定

分析内容：[记录到 SVN\\new4\\07.数据库\\性能调优\\SQL 语句分析结果.txt](#)

内容格式：详见文本内容

举例：对 2009-2-26 问题 1 的分析

=====日期: 2009-2-26=====

问题 1:

详细 SQL 文本：select DF_SID TJXH,BZRQ,PZBH,.....

问题类型： 模块语句问题

(包含： 1、系统语句问题，2、模块语句问题，3、包语句问题，4、无须解决)

优化说明：需要优化

优化后的语句：

分析人： 冯新平

分析日期： 2009-2-27

定位模块： 现金日期账

负责开发小组： F 组

定位人： 刘洪涛 定位日期： 2009-2-27

解决情况： 已解决完，并在 2009-3-1 中进行升级

解决人： 谢新军 解决日期： 2009-3-1

=====END=====

7.2.2 岗位划分

7.2.2.1 问题监控、收集岗

7.2.2.1.1 岗位职责：

1. 安排定制计划任务，每天对数据库进行监控；
2. 对监控到的内容每天 16:00 前通过邮件发送到[语句分析岗](#)。

7.2.2.1.2 责任人：郭建峰

7.2.2.2 语句分析岗

7.2.2.2.1 岗位职责：

1. 将问题监控收集岗发送的内容放到 SVN 上\\new4\07.数据库\性能调优；
2. 对问题最晚在次日内完成分析，并将结果记录在[记录到 SVN\\new4\07.数据库\性能调优\SQL 语句分析结果.txt](#)；
3. 将分析结果通知[模块定位岗](#)；同时，对问题类型为：系统语句问题的结果通知到[问题监控、收集岗](#)，由其自行解决。

7.2.2.2.2 责任人：冯新平

7.2.2.3 模块定位岗

7.2.2.3.1 岗位职责：

1. 根据分析结果，对模块进行定位，以确定问题发生的模块。
2. 每天将定位结果记录在[记录到 SVN\new4\07.数据库\性能调优\SQL 语句分析结果.txt](#)；
3. 将分析结果通知[任务实施岗](#)；

7.2.2.3.2 责任人：刘洪涛

7.2.2.4 任务实施岗

7.2.2.4.1 岗位职责：

1. 根据定位结果，对模块进行修改，并安排到升级记录中。
2. 将修改结果记录在[记录到 SVN\new4\07.数据库\性能调优\SQL 语句分析结果.txt](#)；

7.2.2.4.2 责任人：各开发经理

7.3 技术解决方案

项目日常运行中所遇到的技术上的疑难杂症的解决方案都位于 svn\n4\N4\doc\04 设计实现\0401 解决方案\技术组-1416 目录下，以下只列出主要技术问题。

7.3.1 多事务处理

请参照 svn\n4\N4\doc\01.解决方案/技术组-1416/《VSPN50 关于在实现分布事务处理的解决方法.pdf》

7.3.2 DWR 长连接

请参照 svn\n4\N4\doc\04 设计实现\0401 解决方案\技术组-1416\《DWR 大事物解决办法.txt》

7.3.3 AJAX 同步/异步设置

请参照 svn\n4\N4\doc\04 设计实现\0401 解决方案\技术组-1416\
《ext、buffalo、dwr 同步异步.txt》

7.3.4 ActionForward 事务

请参照 svn\n4\N4\doc\04 设计实现\0401 解决方案\技术组-1416\
《ActionForward 无事务解决方案.txt》

7.3.5 EditorGridPanel 导航

请参照 svn\n4\N4\doc\04 设计实现\0401 解决方案\技术组-1416\
《Ext.grid.EditorGridPanel 导航.txt》

7.3.6 gridpanel 复制

请参照 svn\n4\N4\doc\04 设计实现\0401 解决方案\技术组-1416\
《ext.grid.gridpanel 复制.txt》

7.3.7 权限验证、快捷键、多表头、统计分析条件页、打印

请参照 svn\n4\N4\doc\04 设计实现\0401 解决方案\技术组-1416\《VSPN50 权限验证、快捷键、多表头、统计分析条件页、打印.pdf》

7.3.8 ibatis 分页实现

请参照 svn\n4\N4\doc\04 设计实现\0401 解决方案\技术组-1416\《ibatis 分页实现.txt》