Lodop 复杂图表打印示例

因华通项目中多个模块用到 web 打印，而且部分打印图表较为复杂，以此文做示例和总结。

1 纯 JavaScript 动态生成可变数据行图表

本方案利用 lodop 开放的 api，进行表格的绘制，包括内容和表格线。
适用于结构较简单的可变行图表。

预期效果：

中国华通有限公司

已归库　　入　库　单　　香港车库

入库单号: ffa76900340b0000　供货商 : 新财力胶轮公司　单类别: 入库

入库日期: 2014-04-24　付款方式: 现金　经手人员: 陈燕卿

序号	商品编号	商品名称	型号	类别	单位	数量	币种	原币价	本币价	货位
1	DBDCN200	大巴电池N200		大巴	个	2	HKD	1380	1380	
小计:						0	RMB	0	0	
						2	HKD	2760	2760	

合计金额:

制单人: admin　仓库验收:　主管:　复核:　审批:

分析：
红色部分为可变数据行，行数不固定。
整个表结果，可分表头，表体和表尾，可以看到表头和表位结构固定，变的只是填充的数据而已。
表体，表体的数据行，垂直距离呈有规律递增。
表尾，结构固定，垂直距离随着表体的数据行而变化。

处理思路如下：
定义一个步长定，存表体中的数据行的上边距递增常量。

```
<div id="formTemplate" style="display: none">
</div>

<!-- 出入库单据打印 add by ys on 2014-5-6 -->
<script type="text/javascript">
    var LODOP; //声明为全局变量
    var dataStep = 30;
    //入库单表头左边距，依次为序号，商品编号，商品名称，型号，类别，单位，数量，币种，
    //原币价，本币价，货位(去掉货位)
    var leftPos = [56, 90, 160, 300, 370, 449, 489, 530, 570, 645];
    //出库单表头左边距，依次为序号，商品编号，商品名称，型号，单位，数量，币种，
    //单价，金额，货位，仓库(去掉仓库)
    var leftPosOut= [56, 90, 160, 300, 370, 419, 459, 490, 555, 625];
    var footPos = 0;
```

定义步长

定义一个常量数组，存放表体中数据表头的左边距，本图中有序号，商品编号等 11 个字段，固定好左边距。

```
<script type="text/javascript">
    var LODOP; //声明为全局变量
    var dataStep = 30;
    //入库单表头左边距，依次为序号，商品编号，商品名称，型号，类别，单位，数量，币种，
    //原币价，本币价，货位(去掉货位)
    var leftPos = [56, 90, 160, 300, 370, 449, 489, 530, 570, 645];
    //出库单表头左边距，依次为序号，商品编号，商品名称，型号，单位，数量，币种，
    //单价，金额，货位，仓库(去掉仓库)
    var leftPosOut= [56, 90, 160, 300, 370, 419, 459, 490, 555, 625];
    var footPos = 0;

    //入库单打印函数
```

数据表头左边距

数据写入时，左边距与数据表头的左边距对齐，垂直具体按照行数*步长进行递增处理。

```
    //入库单数据行写入
    function PrintRowDatas_In(rows) {
        var topPos = 210;
        for (var i = 0; i < rows.length; i++) {
            LODOP.ADD_PRINT_TEXT(topPos,leftPos[0], 32, 20, i+1);//写入序号
            LODOP.ADD_PRINT_TEXT(topPos, leftPos[1], 65, 20, rows[i].ProjectCode); //写入商品编号
            LODOP.ADD_PRINT_TEXT(topPos, leftPos[2], 125, 20, rows[i].ProjectName.substring(0,9)); //写入商品名
            LODOP.ADD_PRINT_TEXT(topPos, leftPos[3], 69, 20, rows[i].Version); //写入型号
            LODOP.ADD_PRINT_TEXT(topPos, leftPos[4], 33, 20, rows[i].Category); //写入类别
            LODOP.ADD_PRINT_TEXT(topPos, leftPos[5], 35, 20, rows[i].Unit); //写入单位
            LODOP.ADD_PRINT_TEXT(topPos, leftPos[6], 38, 20, rows[i].Quantity); //写入数量
            LODOP.ADD_PRINT_TEXT(topPos, leftPos[7], 34, 20, rows[i].CurrencyType); //写入币种
            LODOP.ADD_PRINT_TEXT(topPos, leftPos[8], 43, 20, rows[i].Price); //写入原币价格
            LODOP.ADD_PRINT_TEXT(topPos, leftPos[9], 47, 20, rows[i].StandPriceStr); //写入本位价
            //LODOP.ADD_PRINT_TEXT(topPos, leftPos[10], 66, 20, rows[i].Place); //写入货位

            LODOP.ADD_PRINT_LINE(topPos + 15, 55, topPos + 15, 760, 0, 1);
            LODOP.SET_PRINT_STYLEA(0, "Horient", 3);
            topPos = topPos + dataStep;
        }
```

遍历数据项，执行可变行数据写入，步长递增，还有表格线绘制。

表头和表尾的数据写入，可使用 Lodop 的涉及视图，直接拖放控件快速布局（需要 Lodop 的商用授权开启功能），免费版尚不支持。

Demo 代码函数

1 处入库单打印入口函数
2 处绘制表头
3 处绘制表体，写入可变数据行。
4 处绘制表尾。

2 模板生成，可变数据行图表

本方案利用静态模板进行打印内容的绘制，适用于结构较复杂的可变行图表生成。

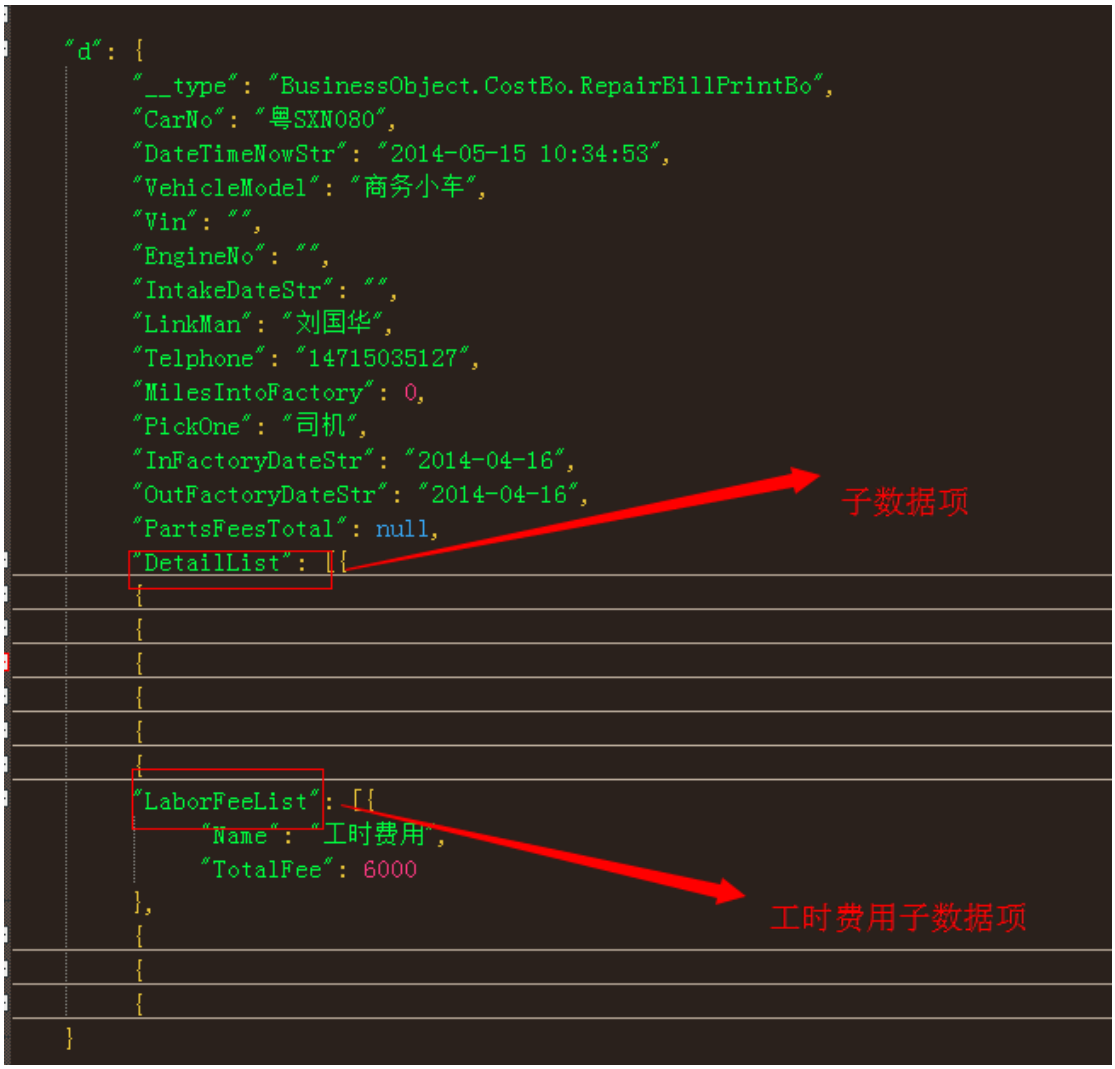
预期效果：

方框圈起来是数据行可变。具体行数根据，返回的数据源中的子数据项决定，即下图中的 detaillist 中的元素个数。

A 区数据，维修单中的工时费用，数据项为 Detaillist 的子项，数目不固定。

B 区数据，维修单中的配件费用，数据项为 Detailslist 中的有效数据子项，注意，本示例中，Detaillist 中存在无效数据项。B 区数据只取 detaillist 中的有效数据项填充。数据项不固定。

看返回的数据源结构



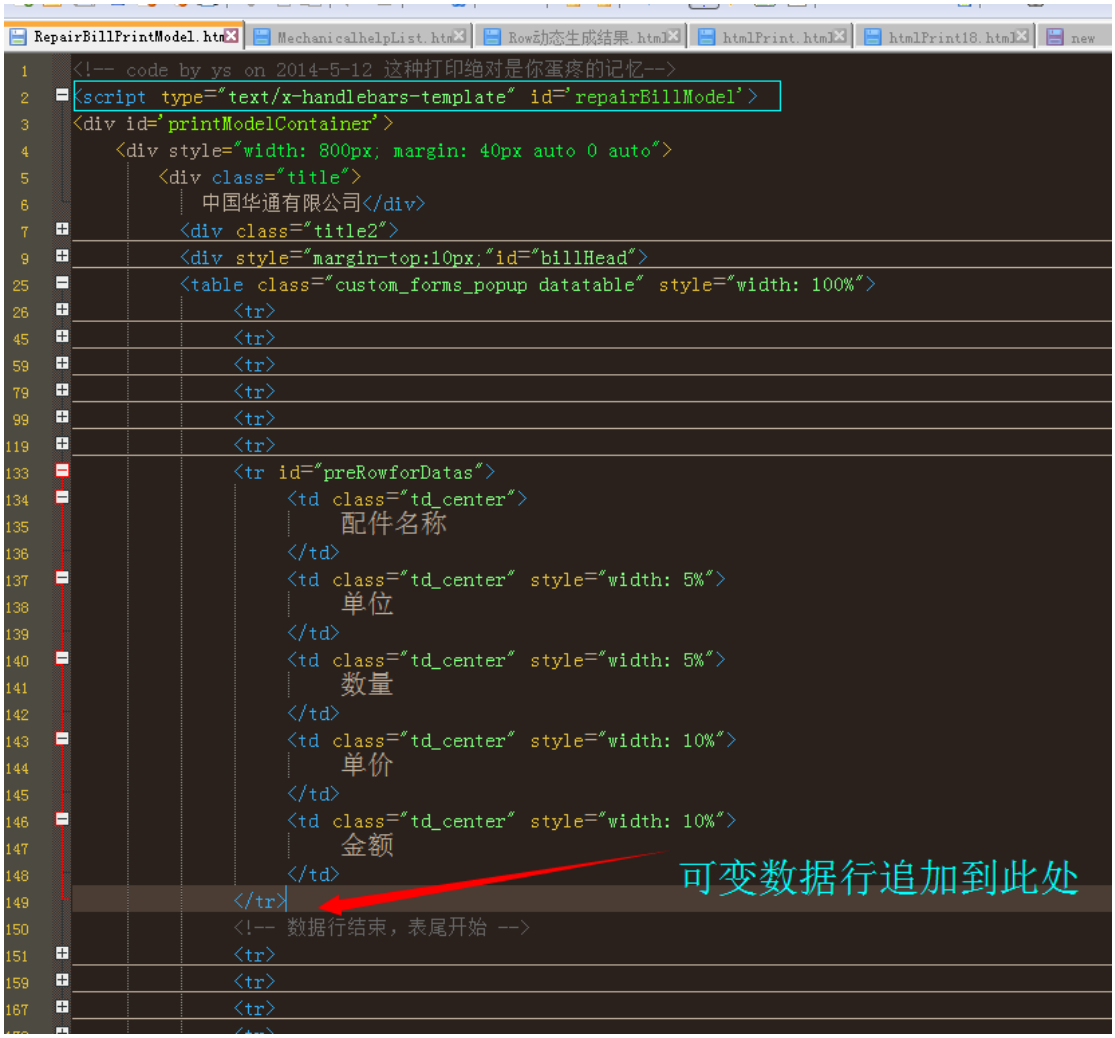
解决思路

采用 lodop 的打印 html 的接口，动态数据行生成(生成 tr，td，填入数据)，追加到 template 中可变数据行的预留位置，然后再针对大表进行其他数据的填充。

对于 A 区的数据，需要再额外做一次数据填充。

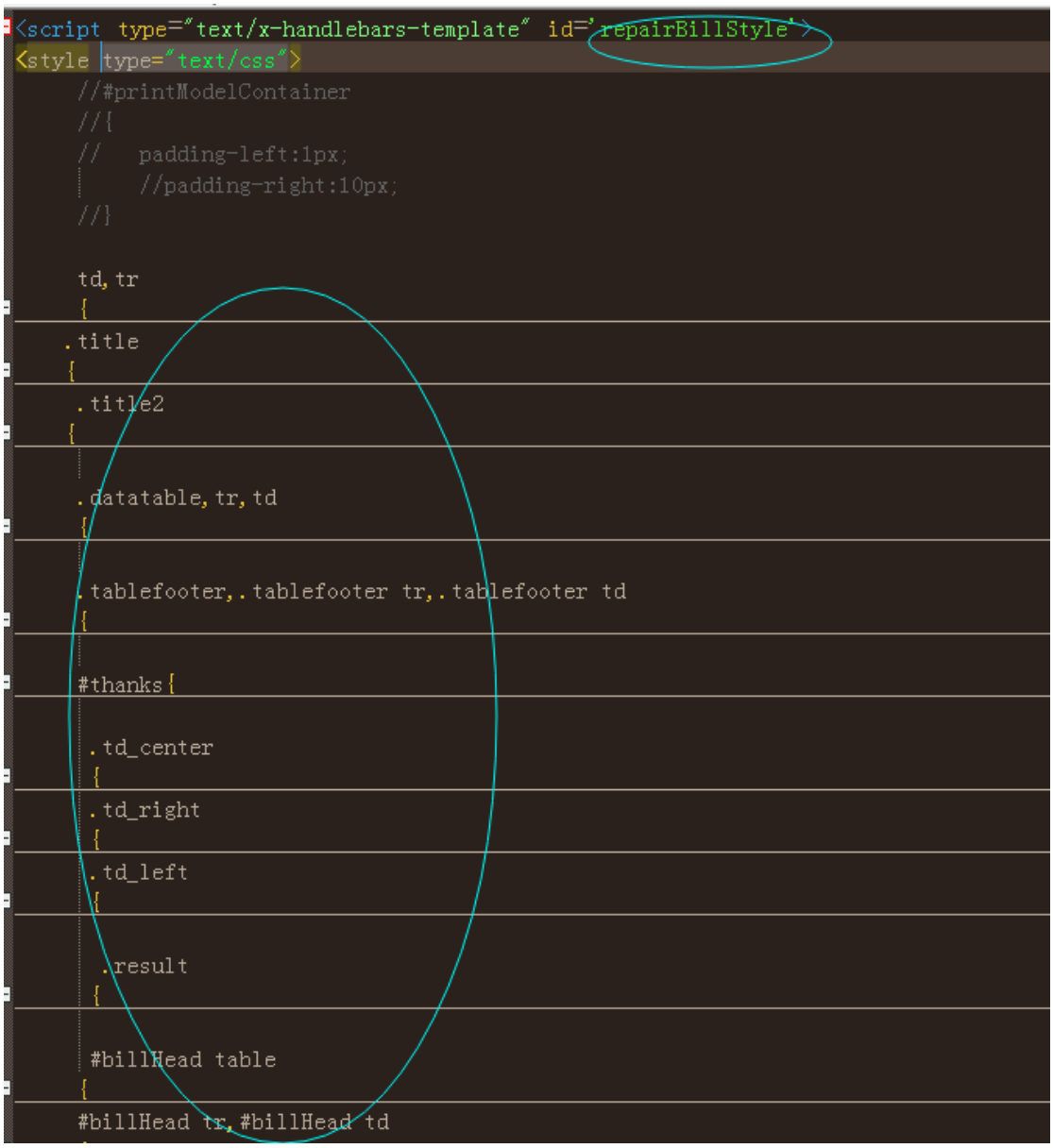
具体如下：

大表结构基本固定，AB 区域可变。定义一个 html 静态模板，可变区域预留。



针对全局表格定义一个模板，表格类容存放于一个容器，本处存放于 Script 标签内，避免 dom 加载的时候就进行 html 的解析。同时也是为了避免对 存在 template 的页面造成干扰。

接下来需要针对打印页面定义样式，样式内容跟 template 一样， 存放于容器中。再取得打印需要的所有 html 之前，将所有样式的 css code string 取出，链接打印内容形成带样式的 html 片段，一起输出打印。



这里存放 css code 的容器也同样采用 Script 标签。避免对 template 的容器页造成干扰。

无视我这烂的不能再烂的 css code 吧。

全局需要的两大块模板准备就绪。可变数据行，可以动态的通过“tr，td”字符拼接。但是我很讨厌这种 string concat 的形式。我也同样采用数据行模板 来生成。

```
<script type="text/x-handlebars-template" id='rowModel' style="display:none;">
    <!--  打印动态数据行模板-->
            <td class="TD_rows td_left" colspan="2" ></td>
          <td class="TD_rows td_center" ></td>
          <td class="TD_rows td_center" ></td>
          <td class="TD_rows">
              {{PartsName}}
          </td>
          <td class="td_center TD_rows">
              {{Unit}}
          </td>
          <td class="td_center TD_rows">
              {{Quantity}}
          </td>
          <td class="td_center TD_rows" >
              {{Price}}
          </td>
              <td class="td_center TD_rows">
              {{Total}}
          </td>
</script>
```

数据行模板定义，也存放于 Script 标签里，相当于文本了。不会模板的容器页面造成干扰，不多说。

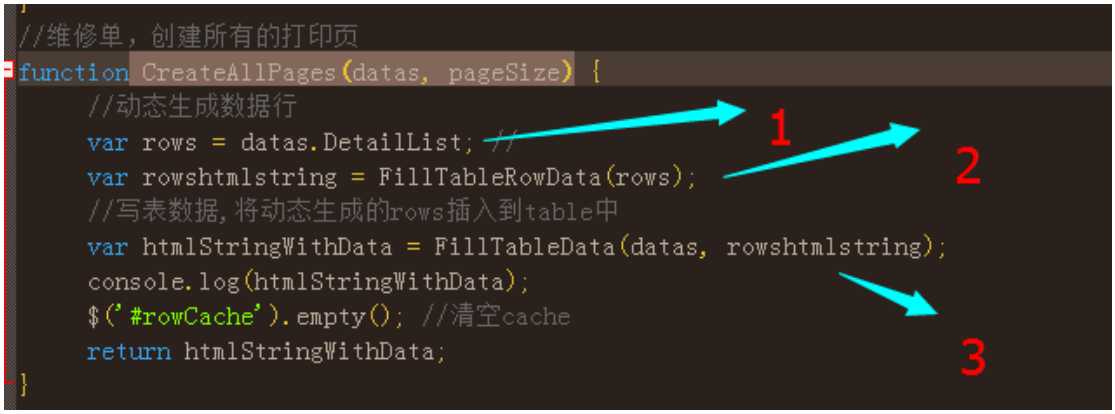
注意{{}}这种使用语法，这个是因为 我使用了模板引擎 handlebar 中的使用语法。用 handlebar 来自动绑定数据。跟 Angular.js 中的语法一样。

准备结束，放代码。

```
//维修单打印函数
function repairBillPrint(datas) {
    if (typeof datas == "undefined")
        return;
    LODOP = getLodop(document.getElementById('LODOP_OB'), document.getElementById('LODOP_EM'));
    LODOP.PRINT_INIT("维修单打印" + Math.random());
    var cssstyle = $("#repairBillStyle").html();
    var printStr = cssstyle + CreateAllPages(datas, 1);
    LODOP.ADD_PRINT_HTM(15, 0, '100%', '100%', printStr);
    LODOP.SET_PRINT_STYLEA(0, "Horient", 2); //设置打印内容居中
    LODOP.SET_PRINT_MODE("PRINT_PAGE_PERCENT", "Auto-Width"); //自动宽度打印
    LODOP.SET_SHOW_MODE("HIDE_PAGE_PERCENT", true);
    LODOP.SET_PRINT_MODE("FULL_HEIGHT_FOR_OVERFLOW", true);
    LODOP.PREVIEW();
    //LODOP.PRINT_DESIGN();
}
```

第一步取样式，第二部，样式和生成的打印页内容连接起来，构成带样式的 html 页面

CreateAllPages(datas, pageSize)函数，创建打印内容页



1， 从数据源中分离出可变数据行需要的数据源，本处是 DetailList

2， 利用 rows 动态生成所有的可变行代码。并装填行数据。

3， 将生成的行添加到大图表的指定位置，并针对图表，进行其他数据装填。

以上的数据装填均采用模板填入，如果数据少，也可手动赋值。

1 装填图表数据

2 将动态数据行代码写入 cache 容器，以便 dom 操作进行节点追加

3 跟具体业务相关，本处工时费项目要求列在左侧栏，循环将工时费数据写入到已经生成的数据行中。

FillTableRowData(rows)函数，生成可变行代码，并填入可变行的行数据

我生成动态行的代码如下。

Tips: 本处示例，几乎等同于“人肉”拼接组装出外面需要的结果，未做性能考虑。而且 huatong 项目中借鉴了 mvc 思想，页面全部是静态 html 页，并无 mvc 框架的支持，无发使用 razor 语法。

另外注意到了前端有一些其他的模板引擎，提供一些类似于 razor 的语法支持，代码更简洁更优雅。如腾讯 CDC 的 artTemplate，阿里系 Kissy Template，百度系 Baidu Template 等
以下演示 artTemlate 的循环语法用法。

```
<h3>
<% if (typeof content === 'string') { %>
        <%= content %>
<% } %>
</h3>
```

其他不做赘述，有需要自行了解。

FillTableRowData(rows)函数返回的是所有生成的动态行(已填入数据，效果如下)，请忽略注释。

```
"<tr class='TR_rows'>
<!-- 打印动态数据行模板--> <!--<td class="td_right TD_rows" colspan="2" ></td>
<td class="td_right TD_rows" ></td>
<td class="td_right TD_rows" ></td>
<td class="td_right TD_rows" > 机油 | </td>
<td class="td_right TD_rows"></td>
<td class="td_right TD_rows"> 2 </td>
<td class="td_right TD_rows" > 160 </td>
<td class="td_right TD_rows"> 320 </td>--> </tr>
<tr class='TR_rows'>
<!-- 打印动态数据行模板-->
<!--<td class="td_right TD_rows" colspan="2" ></td>
<tr class='TR_rows'>
<!-- 打印动态数据行模板-->
<!--<td class="td_right TD_rows" colspan="2" ></td>
<td class="td_right TD_rows" ></td>
<td class="td_right TD_rows" ></td>
<td class="td_right TD_rows" > 空调格 | </td>
<td class="td_right TD_rows"></td>
<td class="td_right TD_rows"> 1 </td>
<td class="td_right TD_rows" > 60 </td>
<td class="td_right TD_rows"> 60 </td>--> </tr>
<tr class='TR_rows'>
<!-- 打印动态数据行模板-->
<!--<td class="td_right TD_rows" colspan="2" ></td>
<td class="td_right TD_rows" ></td>
<td class="td_right TD_rows" ></td>
<td class="td_right TD_rows" > 换机油、机油格。空调格 </td>
<td class="td_right TD_rows"></td>
<td class="td_right TD_rows"> 1 </td>
<td class="td_right TD_rows" > 60 </td>
<td class="td_right TD_rows"> 60 </td>--> </tr>"
```

FillTableData(datas, rowshtmlstring)函数，追加生成的行代码，分离出工时费数据项，向已生成的数据行中追加工时费数据项，并装填其他图表数据

```
//写入维修项目（工时费）
var laborFees = datas.LaborFeeList; //
for (var j = 0; j < laborFees.length; j++) { //遍历维工时费数据项
    var tempFee = laborFees[j];
    var createdRows = $('.CreatedRow '); //找出所有动态创建的行
    var tempRow = createdRows[j]; //按照序号取出动态创建的行，填入对应序号的数据
    var tdFeeName = tempRow.cells[0]; //装填name的td
    var tdFeeTotal = tempRow.cells[2]; //装填total的td
    if (typeof tdFeeName != "undefined" && typeof tempFee.Name != "undefined") {
        $(tdFeeName).text(tempFee.Name); //写入维修项目名称
    }
    if (typeof tdFeeTotal != "undefined" && typeof tempFee.TotalFee != "undefined") {
        $(tdFeeTotal).text(tempFee.TotalFee); //写入维修项目费用
    }
}
var allhtml = $('#rowCache').html();
//console.log(allhtml);
return allhtml;
```

1 处分离出工时费数据项

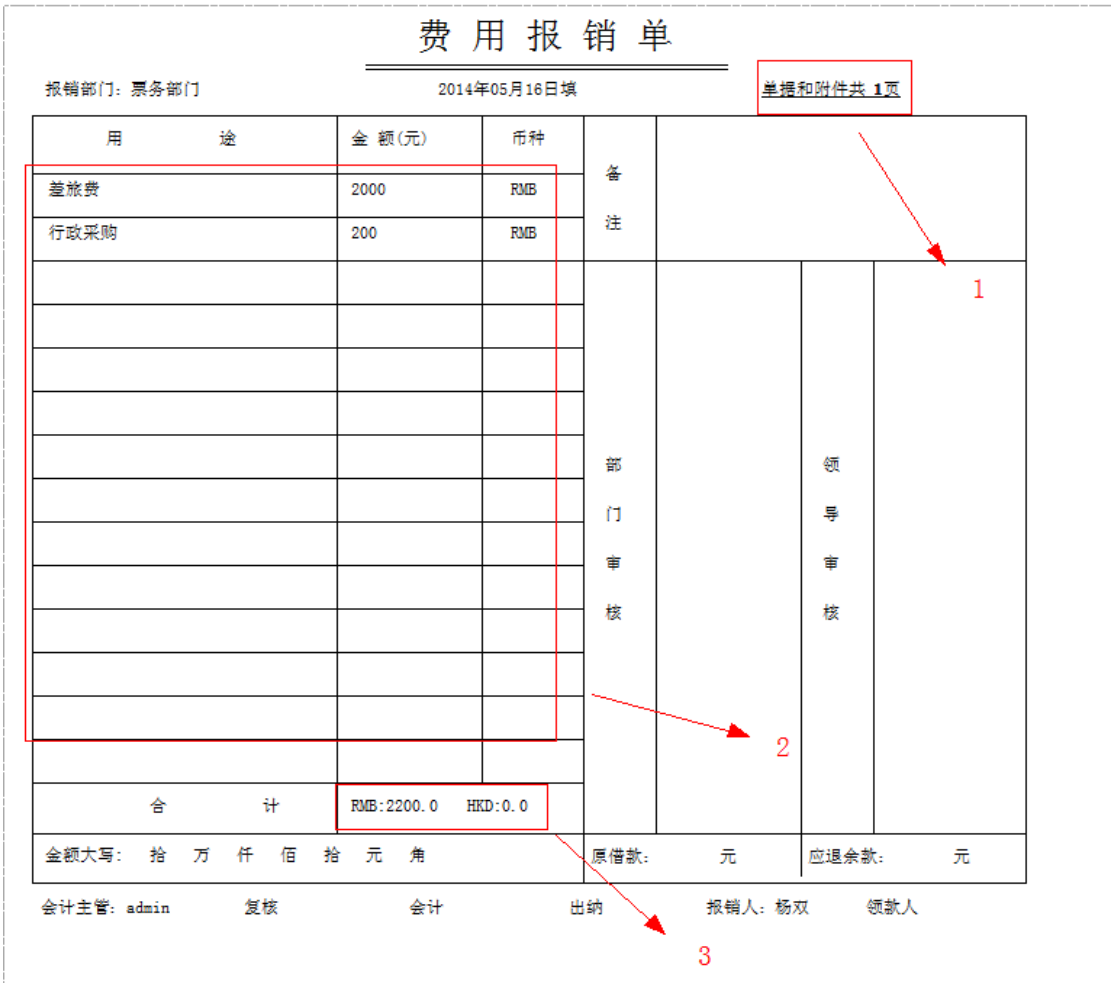
2 处，对数据项遍历，写入数据
3 处，将生成的 html 放入缓存容器，以便能利用 Jquery 进行操作。

到这里，利用模板进行复杂图表的生成介绍结束。

3 分页图表生成和打印

这部分，讲解带分页的图表打印。虽然 Lodop 有一些 api 提供了分页打印，但是，那种分页支持不能满足要求。

期望效果如下：



导致有分页需求的， 是如果有多达几十项的报销子项。那么需要打印这种表格应该是打印多张。每一张，表头和表尾巴内容是相同，每一页变动的是表体内容，结构不会发生变化。

1 处是页码
2 处报销子项，写入了项目和币种和金额。

3 处，每一页的报销单小计求和。

针对这种需求的打印。变的是数据，结构不便。很显然，最优的做法是利用上面第二种方案。模板化处理，再打印。

很不幸，一开始，这个图标的打印， 我是完全用脚本写的。包括表格线的绘制，维护起来很困难。

最优做法采用 template 来做，参考第二种方案的实现思路。本处仍然按照脚本生成来说明。

这个分页打印，稍微麻烦点的就在分页，其他都很简单，针对一条数据，打印详情，分页必然要做假分页，看下代码。

1 处，pageSize 大小
2 处，计算页数
3 处，真正的打印入口处理函数

CreateAllPages(datas, pageSize)函数，传入所有数据和分页大小。这是分页打印中最关键的一个方法

```
//创建所有的打印页
function CreateAllPages(datas, pageSize) {
    var counts = datas.Pages;//取打印页数
    var totalRows = datas.DetaisPrint; //取所有的打印数据
    for (var i = 1; i<= counts; i++) {
        LODOP.NewPage();
        PrintTable(datas); //创建表结构
        var pageRows = GetCurrentPrintPageRows(totalRows, i, pageSize);//注意pageRows最末索引未知的值为单页打印表格的统计值
        MyDataPreView(pageRows, i, counts); //写入分页数据
        var totalPriceRMB = pageRows[pageRows.length - 2];
        var totalPriceHKD = pageRows[pageRows.length - 1];
        //console.log('totalPriceRMB' + totalPriceRMB);
        //console.log('totalPriceHKD' + totalPriceHKD);
        var a = pageRows.pop();
        console.log('pop' + a);
        fillPageApplyTotalPrice(parseFloat(totalPriceRMB).toFixed(1),parseFloat(totalPriceHKD).toFixed(1));//写入报销单的总价
    }
};
```

1 处取出打印页总数
2 取出打印数据子项集合
3 创建新打印视图页，并绘制表结构
4 获取分页数据源
5 执行分页数据的写入，和分页小计的统计
6 写入分页小计数据

获取分页数据源函数处理 GetCurrentPrintPageRows(totalRows, index, pageSize)如下

```
//获取打印用的分页据数，并且返回统计小计，分币种统计
function GetCurrentPrintPageRows(totalRows, index, pageSize) {
    var returnrows = [];
    var totalPrice = 0;//人民币总价
    var totalPriceHKD = 0; //港币总价
    var counts = 0;
    var pages = totalRows.length / pageSize; //取整数页
    var intcount = parseInt(pages);
    if (pages > intcount) {
        counts = intcount + 1;
    }
    else {
        counts = intcount;
    }
    var beginpointer = (index - 1) * pageSize; //数据指针'
    var endpointer = index * pageSize; //数据指针

    for (var i = beginpointer; i < endpointer; i++) {
        var tempData = totalRows[i];
        if (tempData != null && typeof (tempData) != 'undefined') {
    }
    totalPrice = totalPrice.toFixed(1); //有效数字修正
    totalPriceHKD = totalPriceHKD.toFixed(1); //有效数字修正
    var tempStrRMB = totalPrice.toString();
    var tempStrHKD = totalPriceHKD.toString();
    returnrows.push(tempStrRMB); //写入一页报销单的人民币总价
    returnrows.push(tempStrHKD);//港币总价
    return returnrows;
}
```

分页数据写入函数 MyDataPreView(rows,pageCountTotal)代码如下

```
//分页数据行写入
function MyDataPreView(rows,pageCountTotal) {
    var nameLeft = 30; //数据循环打印，左边距从120px开始显示写入报销项目
    var nameTop = 120; //上边距从120px开始写入数据
    var priceLeft = 240; //金额的左边距
    var currencyLeft = 350;
    var dataStep = 30; //数据写入在垂直方向上的步长
    for (var i = 0; i < rows.length-1; i++) {
        LODOP.ADD_PRINT_TEXT(nameTop, nameLeft, 100, 20, rows[i].ItemName);
        LODOP.ADD_PRINT_TEXT(nameTop, priceLeft, 60, 20, rows[i].Price);
        LODOP.ADD_PRINT_TEXT(nameTop, currencyLeft, 30, 20, rows[i].Currency);
        LODOP.ADD_PRINT_TEXT(51, 600, 79, 20, pageCountTotal);//写入总页数
        LODOP.SET_PRINT_STYLEA(0, "Underline", 1);
        nameTop = nameTop + dataStep;
    }
}
```

处理手段类似，要定义垂直距离步长

Tips：带分页的打印，类似于上述打印效果的，不要使用脚本生成，麻烦，而且难以维护。用 template 处理吧。

三种结构动态的较为复杂的图表打印介绍完了，对于其他一些结构静态的图表，不会很难，参考 lodop 的文档，照猫画虎。

有问题的私下找我询问。

By ys on 2014-5-16