Smartbi新增图形类型指南

该文档以添加D3柱图为例，说明如何在smartbi中新增图形控件。

示例代码下载　D3Chart.rar

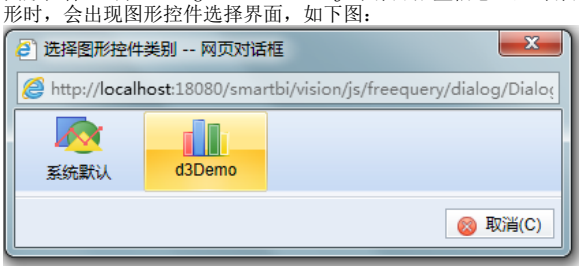
文档目录：

一、前端设置界面

该步骤主要涉及前端图形设置界面实现，添加新的图形配置入口及界面展示。

1、图形控件选择入口

Smartbi内置支持funsionCharts和highcharts图形控件，如果要增加新的图形控件，可在ConfigurationPatch.js中添加配置信息，此时添加新图形时，会出现图形控件选择界面，如下图：



配置详细信息参考ConfigurationPatch.js：

```
var ConfigurationPatch = {
    extensionPoints: {
        css : [ '/vision/css/d3Chart.css' ],
        chartPlugIn:[{   扩展图形配置入口
            id:"d3Demo",
            name:"d3Demo",
            iconUrl:"img/chart/charttype/column_group.png",//显示图片
            chartTemplatesClassName:"smartbi.d3chart.ChartTypeTemplates",   //图形模板文件
            defaultChartType:"d3column",
            chartConfig: [//图形配置信息插入点
                {
                    type:"d3column",//图形类型，对应ChartTypeTemplates.js文件中templates对象的type
                    chartConfigClassName:"smartbi.d3chart.ColumnChartConfig",//界面设置配置文件路径
```

图形控件选择界面信息

2、图形配置界面实现

Smartbi 最新版本支持2种方式实现图形配置界面，一是同产品功能紧密结合的配置文件方式，遵循Smartbi框架规范，继续使用原有的图形设置界面；二是配置界面完全使用独立的页面实现，通过html独立开发完成 。建议使用第二种　**独立页面的实现方式**，不强制遵循smartbi的界面规范，比较简单灵活容易实现。

2.1 独立页面方式实现图形配置界面

```
var ConfigurationPatch = {
    extensionPoints: {
        chartPlugIn:[{
            id:"ECharts",
            name:"ECharts",
            iconUrl:"img/chart/charttype/echarts.png",//显示图片
            chartTemplatesClassName:"smartbi.echarts.ChartTypeTemplates",    //图形类型选择器模板文件
            defaultChartType:"echarts_bar",//默认图形类型
            showSelectChartTypeTemplates:true,  //弹出图形模块选择界面
            //chartEditUrl:"test/customEditor.html",     //图形自定义编辑界面，需要在页面中调用回调函数。
            chartConfig: [//图形配置信息插入点
                {
                    type:"echarts_bar",//图形类型，对应ChartTypeTemplates.js文件中templates对象的type
```

选择图形控件后，会自动打开用户设置的页面，并传递相关参数信息，用户在关闭页面后，需要返回相关信息。

参考示例：以下为customEditor.html 文件头的代码示例，主要参考该页面中js接受参数信息及如何返回配置信息。

```
<script>
	var args = window.dialogArguments;
	if (!args && window.opener && window.opener._dialogArguments) {
		args = window.opener._dialogArguments;
		try {
			delete window.opener._dialogArguments;
		} catch (e) {
		}
	}
	var data, fn, obj, win;
	function closeWindows() {
		var returnValue = {
			chartType:"echart_test",//optionbuild
			chartDefine:{},//chartDefineoptionbuild
			externalDefine:{}//optionbuild
		};
		//baseDialog.close(returnValue);
		if (fn) {
			try {
				fn.apply(obj, [ returnValue ]);//
			} catch (e) {
			}
		}
		window.close();
	}
	function init() {
		//
		data = args[2], fn = args[3], obj = args[4], win = args[5];
		//baseDialog.init(data, fn, obj, win);
	}
	function destroy() {
	}
</script>
```

2.2产品默认配置文件方式

2.2.1 图形类型选择器

选择图形控件后，需要继续选择图形类型及展现配置界面。smartbi的图形配置界面分为2部分展示，效果如下：

其中上半部分图形类型选择界面，一种图形控件的所有图形类型都是相同的。下半部分具体图形类型详细配置界面，不同的图形类型，配置界面可能都不同。
该步骤是实现图形类型选择界面信息。

1）添加入口

在ChartTypeTemplates.js添加图形类型选择器入口

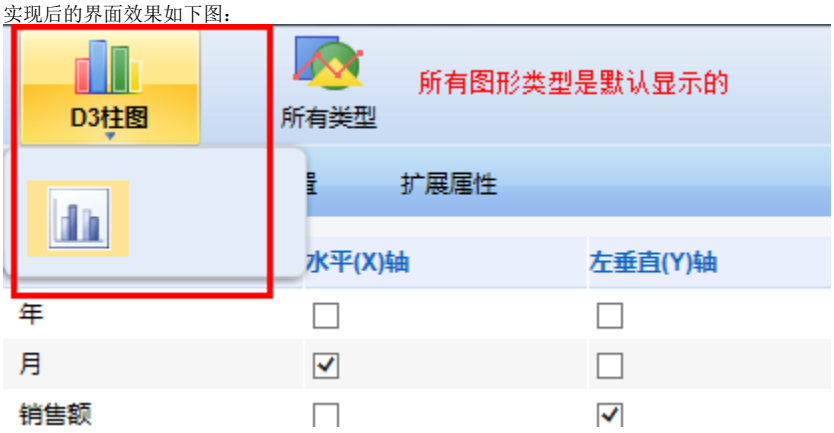
```
chartPlugIn:[{
    id:"d3Demo",
    name:"d3Demo",
    iconUrl:"img/chart/charttype/column_group.png",//显示图片
    chartTemplatesClassName:"smartbi.d3chart.ChartTypeTemplates",    //图形类型选择器模板文件
    defaultChartType:"d3column",//默认图形类型
    chartConfig: [//图形配置信息插入点
        {
            type:"d3column",//图形类型，对应ChartTypeTemplates.js文件中templates对象的type
            chartConfigClassName:"smartbi.d3chart.ColumnChartConfig",//界面设置配置文件路径
            chartClassName:"smartbi.d3chart.D3Chart",//图形展现使用配置文件路径
            chartViewClassName:"smartbi.d3chart.D3ChartView"//图形前端展现配置文件路径
        }
    ]
}]
```

2）实现图形类型选择界面

实现步骤2.1中配置的ChartTypeTemplates文件；该文件可直接继承AbstractChartTypeTemplates.js,需要实现的方法有2个：initTemplatesGroup和initTemplates。
详细内容可参考\js\smartbi\d3chart\ChartTypeTemplates.js。

```
var ChartTypeTemplates = AbstractChartTypeTemplates.extend({

/**
* 初始化模板分组信息
* @returns 图形分组模板对象，对象信息如下：
* key：唯一key值，对象具体信息，内容如下：
* name：模板显示名称
* iconUrl：显示图片地址
* templateKeys，数组对象，包含的templates图形类型,initTemplates中的对象key

*/
initTemplatesGroup:function(){
this.templatesGroup = {
D3:{
name:'D3柱图',
iconUrl:'img/chart/charttype/column_group.png',//图片地址
templateKeys:['d3column']//包含的templates图形类型,此处为新加d3column类型
}
}
return this.templatesGroup;
},

/**
* 初始化模板包含的图形类型对象，
* @returns 模板对象，对象包括：
* 对象key、唯一key值：对象内容，内容是数值对象，包括如下信息：
* id、唯一id
* name、唯一名称
* type、唯一图形类型
* iconUrl、图片地址
* options 默认设置，可以为空。
* @returns 图形类型模板对象
*/
initTemplates:function(){
this.templates = {
d3column : [ {
id : 'd3column',//唯一的key。
name : 'd3柱图',
type : 'd3column',//图形类型
iconUrl : 'img/chart/charttype/column_common.png',
options : {}
} ]
}
return this.templates;
}

});
```

实现后的界面效果如下图：



2.2.2 具体图形类型配置文件入口

点击界面上新加的图形类型后，需要找到对应的配置文件，以加载图形配置界面。
在ConfigurationPatch.js中添加配置信息。可参考ConfigurationPatch.js；如下图

```
chartPlugIn:[{
    id:"d3Demo",
    name:"d3Demo",
    iconUrl:"img/chart/charttype/column_group.png",//显示图片
    chartTemplatesClassName:"smartbi.d3chart.ChartTypeTemplates",    //图形类型选择器模板文件
    defaultChartType:"d3column",//默认图形类型
    chartConfig: [//图形配置信息插入点   图形控件需要实现的图形类型及相关配置入口
        {                                图形类型及对应的图形设置界面实现文件
            type:"d3column",//图形类型，对应ChartTypeTemplates.js文件中templates对象的type
            chartConfigClassName:"smartbi.d3chart.ColumnChartConfig",//界面设置配置文件路径
            chartClassName:"smartbi.d3chart.D3Chart",//图形展现使用配置文件路径
            chartViewClassName:"smartbi.d3chart.D3ChartView"//图形前端展现配置文件路径
        }
    ]
}]
```

2.2.3 实现设置界面的配置文件

即实现步骤3中配置chartConfigClassName对应的文件，该文件是用来展现图形设置界面的，分为总体配置和详细配置

2.2.4 总体配置实现

1. 参考smartbi.d3chart.ColumnChartConfig.js。
2. 需要继承抽象类：smartbi.chart.interface.AbstractChartConfig.js
3. 该文件需要整合图形设置界面的功能项，并实现getChartConfig()方法，返回一个json对象；产品会根据返回内容自动生成tab页签，如下图：


```
/**
 */
ColumnChartConfig.commonChartConfig = {
    chartDatas : {
        key : "chartDatas",//数据设置是必须的，并且key是固定的
        name : "数据设置",
        //数据设置必须有对应的className，对应一个js文件,该文件用来处理图形与产品数据集的关联关系
        className : "smartbi.d3Chart.setting.DataSetSetting"
    },
    basicSetting : {
        fakeKey : "basicSetting",
        name : "基本设置",
        items : []
    },
    other : {
        key : "other",
        name : "扩展属性",
        className : "smartbi.chart.setting.settingitem.OtherSetting"
    }
};
ColumnChartConfig.commonChartConfig.basicSetting.items[0] = BasicSetting.items;

/**
 */
ColumnChartConfig.getChartConfig = function(chartType) {
    var newChartConfig = jQuery.extend(true, {}, ColumnChartConfig.commonChartConfig);
    return newChartConfig;
}
```

4. 返回对象中必须有一个key为chartDatas的属性，配置图形与数据集的关联关系。

2.2.5 详细配置实现

实现"总体配置"中每个tab页的功能。

1） 实现数据设置功能

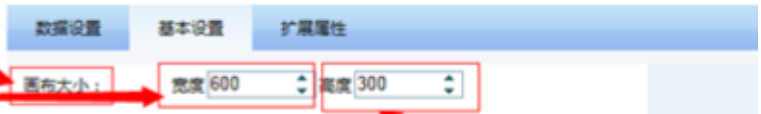
1. 数据设置是必须的，并且要继承抽象类AbstractDataSetSetting.js,参考DataSetSetting.js
2. 示例中直接使用抽象类的所有实现，展现效果如下：


```
var DataSetSetting = function(container, settingControl) {
    this.container = container;
    this.settingControl = settingControl;
    this.chartType = settingControl.getChartType();
    this.reportType = settingControl.reportType;

    this.key = "chartDatas";
    this.subKey = "fieldList";
    this.needRefreshDemo = true;
    this.needRefreshDemoEvent = new CustomEvent("needRefreshDemoEvent");
    this.init(container, __url);
    this.render();
}

lang.extend(DataSetSetting, "smartbi.chart.abstract.AbstractDataSetSetting");

DataSetSetting.prototype.destroy = function() {
    DataSetSetting.superclass.destroy.call(this);
}
```

2）实现基本设置功能

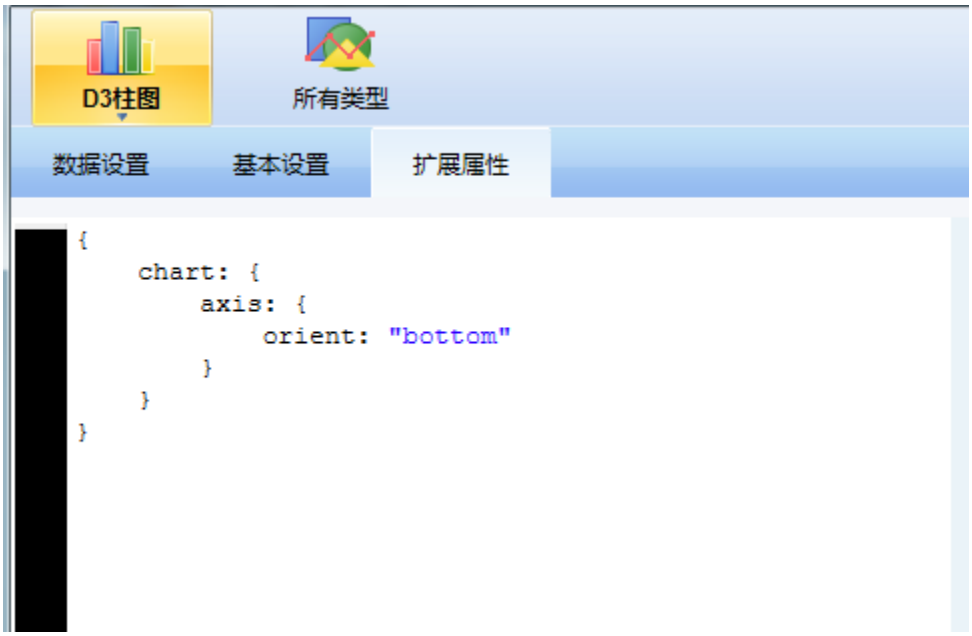
1. 该tab页不是必须的，针对不同的图形，可自定义属性配置项。参考BasicSetting.js
2. 示例中展现效果如下：

```
//基本设置项
var BasicSetting = {};
BasicSetting.items = {
    key : "chart",
    name : "画布大小",
    type : "rowGroup",
    items : [ {
        key : "width",
        name : "宽度",
        control : "SpinnerControl",
        controlAttribute : {
            minValue : 0,
            maxValue : 3000,
            width : 80,
            defaultValue : 600
        }
    },{
        key : "height",
        name : "高度",
        control : "SpinnerControl",
        controlAttribute : {
            minValue : 0,
            maxValue : 2000,
            width : 80,
            defaultValue : 400
        }
    } ]
```



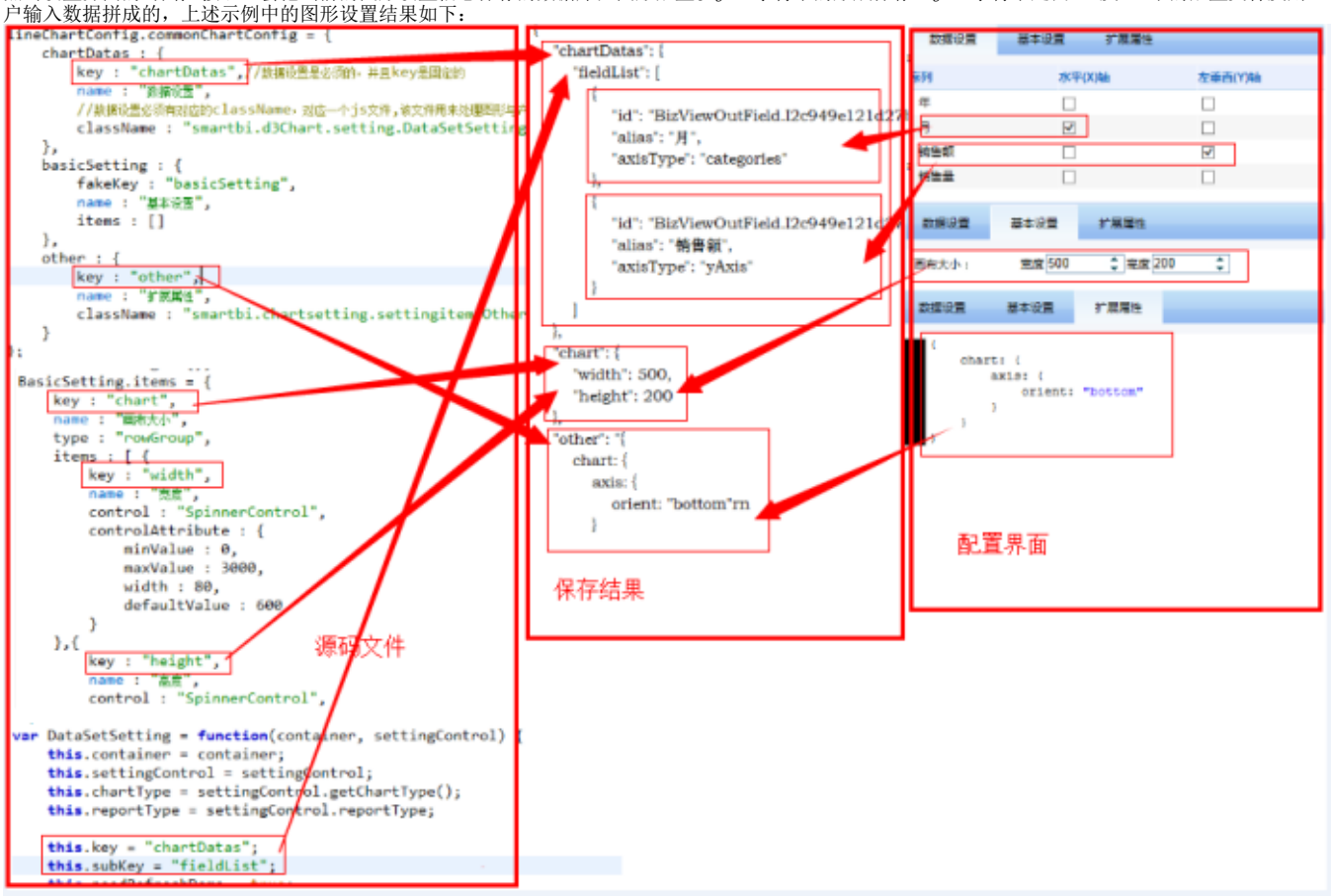
3）实现扩展属性功能

1. 该tab页不是必须的，示例中直接引用产品已有的实现
2. 参考smartbi.chartsetting.settingitem.OtherSetting，该功能支持直接写json格式的图形属性，浏览图形是会自动把代码合并到图形属性中；界面显示如下：



2.2.6 保存图形配置

点击设置界面的"保存"按钮，会把当前的图形设置信息保存的数据库；图形配置以json字符串的形成保存，json字符串是由3.1及3.2中的配置文件及用户输入数据拼成的，上述示例中的图形设置结果如下：



二、后台生成图形对象逻辑

该步骤主要是根据保存在数据库中的图形设置信息，及关联的数据字段信息，生成图形控件可以接受的对象。

1、图形后端处理类

实现接口IChartOptionsBuilder.java。
该类主要功能是根据存放在数据库中的图形设置信息及使用的数据集数据，生成可以给客户端图形控件直接使用的对象。可参考D3LineOptionsBuilder.java（以继承AbstractOptionsBuilder为例）。
产品本身已经实现了2种图形控件类型，funsionCharts和highcharts，如果新加图形属于以上2种控件，可直接继承BaseOptionsBuilder.java
(funsionCharts基本实现)
或HighChartsBaseOptionBuilder.java(highcharts基本实现)
接口文件内容如下：

```
/**
* 根据图形设置定义，生成客户端js创建图形的图形选项。
*
* @author ms 创建时间:2012-3-21
*/
public interface IChartOptionsBuilder {
/**
* 生成 客户端js图形接收图形定义。
*
* @param chartDefine
* 图形定义所对应的实体类 （数据库对象），系统缺省使用的图形定义类
* @param data
* 图形关联数据集对应的数据
* @return JSONObject
*/
JSONObject build(ChartDefine chartDefine, BaseChartData data);

/**
* 生成 客户端js图形接收图形定义。
* @param chartDefine
* 图形定义所对应的实体类 （数据库对象），系统缺省使用的图形定义类
* @param data
* 图形关联数据集对应的数据
* @param width
* 重新定义的图形宽度
* @param height
* 重新定义的图形高度
* @return JSONObject 客户端js图形接收图形定义
*/
JSONObject build(ChartDefine chartDefine, BaseChartData data, int width, int
height);

/**
* 判断该Builder类是否能build该类型的图形
*
* @param chartType
* 图形类型
* @return boolean
*/
boolean accept(String chartType);

/**
* 特殊设置项"图片模式"需要实现；否则可以不处理
* 只返回基本的图形配置信息;
* 处理chartOptions.chart下属性
* @param chartDefine 图形设置
* @return 图形配置Json模式。
*/
JSONObject processChartAttrbsOnly(ChartDefine chartDefine);
}
```

2、工厂处理类

实现接口IChartOptionsBuilderFactory。
该类是根据不同的图形类型，获取步骤1中创建的图形处理类（IChartOptionsBuilder实例的接口）。参考D3ChartOptionsBuilderFactory.java
接口文件如下：

```
/**
* 获取 @link IChartOptionsBuilder实例的接口
*
* @author ms 创建时间:2012-4-1
*/
public interface IChartOptionsBuilderFactory {
/**
* 是否接受指定图形类型
*
* @param chartType
* 图形类型
* @return boolean
*/
boolean accept(String chartType);

/**
* 获取指定图形类型的处理类
*
* @param chartType
* 图形类型
* @return 指定图形类型的处理类
*/
IChartOptionsBuilder getChartOptionsBuilder(String
chartType);
}
```

3、注册图形实现类

在服务器启动时，注册新的图形工厂处理类到ChartModule中，以便新加的图形刷新时可以找到服务器实现类。
参考：D3ChartModule.java
调用IChartModule接口的registerChartOptionsBuilderFactory方法，注册步骤2中创建的工厂处理类。

```
14 public class D3ChartModule implements IModule{
15
16     private static D3ChartModule instance;
17
18     private IChartModule chartModule;
19
20     public static D3ChartModule getInstance() {
21         if (instance == null)
22             instance = new D3ChartModule();
23         return instance;
24     }
25
26     public void activate() {
27         //注册图形处理工厂实现类
28         chartModule.registerChartOptionsBuilderFactory(D3ChartOptionsBuilderFactory.getInstance());
29
```

4、测试图形后端生成结果

完成上述步骤后，即可生成图形对象。

三、图形显示及预览

根据图形配置信息，由后端生成图形组件可以接受的对象后，需要客户端做相应处理，把图形展现出来。
说明：

1. 前端图形显示涉及3个文件：ChartView 、serverChart 及Chart ；图形导出时会使用serverChart 和Chart 。
2. 其中ChartView继承serverChart，serverChart引用Chart对象。
3. ChartView主要处理浏览器展现图形时的一些特殊情况，比如点击事件等；它需要继承抽象父类AbstractChartView并实现相关方法。
4. Chart是图形展现的最终处理类，比如调用对应的图形组件画图等；它需要继承抽象父类AbstractChart并实现相关方法。
5. serverChart作为后端导出引用的固定文件，一般不做修改。
6. ChartView和Chart都可以根据图形类型动态创建不同的实现文件。

1、实现图形展现

1.1 配置图形类型对应的展现文件：

ChartView和Chart的实现类，参考ConfigurationPatch.js

```
chartPlugIn:[{
    id:"d3Demo",
    name:"d3Demo",
    iconUrl:"img/chart/charttype/column_group.png",//显示图片
    chartTemplatesClassName:"smartbi.d3chart.ChartTypeTemplates",    //图形类型选择器模板文件
    defaultChartType:"d3column",//默认图形类型
    chartConfig: [//图形配置信息插入点
        {
            type:"d3column",//图形类型，对应ChartTypeTemplates.js文件中templates对象的type
            chartConfigClassName:"smartbi.d3chart.ColumnChartConfig",//界面设置配置文件路径
            chartClassName:"smartbi.d3chart.D3Chart",//图形展现使用配置文件路径
            chartViewClassName:"smartbi.d3chart.D3ChartView"//图形前端展现配置文件路径
        }
    ]
}]
```

1.2 实现ChartView

1. 继承抽象类AbstractChartView
2. chartView主要是处理前端展现和导出时的特殊情况，比如鼠标移上图形时可以显示一个菜单等。如无特殊设置，可以全部使用抽象父类的实现。
3. 由于chart.js在导出时使用，不能动态引入js文件，所以展现图形需要的js文件，都要在chartView文件中引入。要注意引入文件的顺序。

```
if (typeof jQuery == 'undefined' || !jQuery) {
    jsloader.resolve('thirdparty.jquery.jquery', true);
}

if (typeof d3 == 'undefined') {
    jsloader.resolve("d3.d3", true);
}

if (typeof Class == 'undefined') {
    jsloader.resolve("freequery.lang.Class", true);
}

if (typeof AbstractChart == 'undefined') {
    jsloader.resolve("smartbi.chart.AbstractChart", true);
}

if (typeof D3Chart == 'undefined') {
    jsloader.resolve("smartbi.d3chart.D3Chart", true);
}

var AbstractChartView = jsloader.resolve("smartbi.chart.abstract.AbstractChartView");



/**
 * 前端展现逻辑
 */
var D3ChartView = AbstractChartView.extend({
    /**
     * 获取演示数据定义，在图形配置界面刷新数据时用到。
```

1.3 实现Chart

1. Chart是图形最终展现的实现文件，主要根据图形设置信息，调用图形控件对象，把图形展现在浏览器上。
2. 需要继承并实现文件，抽象类方法说明如：

```
/**
```

```
* 图形渲染的接口类，任何新添的图形类型都应该实现该接口
*
* <pre>
* 依赖脚本：
* @freequery.lang.Class;
* </pre>
*/
var AbstractChart = Class.extend({
init : function(chartType, className) {
this.chartType = chartType;
this.className = className;
},

/**
*
*/
destroy : function() {
delete this.chartType;
delete this.className;
},

/**
* 返回图形类型
*
* @returns chartType
*/
getChartType : function() {
return this.chartType;
},

/**
* 设置图形类型
*
* @param chartType
*/
setChartType : function(chartType) {
this.chartType = chartType;
},

/**
* 返回底层的图形对象，如果使用的是highcharts，则是Highcharts.Chart对象
* 返回底层的图形对象，如果使用的是地图，则是Highcharts.Map对象
* @returns chartObject
*/
getChart : function() {
return null;
},

/**
* 返回图形配置信息
*
* @returns options
*/
getOptions : function() {
return null;
},

/**
* 返回图形的svg字符串
*
* @returns svg
*/
getSVG : function() {
return null;
},

/**
* 刷新图形
*
* <pre>
* @param container
* 图形容器，可以是id或dom对象
* @param chartOptions图形配置，
* 如果使用的是highcharts，则是new Highcharts.Chart(options)中options的超集

* 如果使用的是地图，则是new Highcharts.Map(options)中options的超集
* </pre>
*/
refresh : function(container, chartOptions) {
},

/**
* 重设图形到指定大小
*
```

```
* <pre>
* @param width
* 新的图形宽度
* @param height
* 新的图形高度
* </pre>
*/
resizeTo : function(width, height) {
},

/**
* 触发图形点击事件
* <pre>
* @param pointObj 点击对象的属性信息等
* </pre>
*/
pointClick : function(pointObj,arg1,arg2,arg3) {
var e = window.event;
this.fire('PointClick',pointObj, this,e);
},

/**
* 触发图形BeforeRenderer事件,在加载或刷新图形之前执行
* <pre>
* </pre>
*/
onBeforeRenderer : function() {
this.fire('BeforeRenderer',this);
},

/**
* 触发图形AfterRenderer事件,在刷新图形之后执行
*/
onAfterRenderer : function() {
this.fire('AfterRenderer',this);
}
});
```

3. 具体实现可参考smartbi.d3chart.D3Chart.js

四、宏代码相关

1. 图形目前有3个宏事件，beforeRender和afterRender事件，以及click事件。
2. beforeRender事件主要用来修改图形属性信息，只要添加触发点就可以了。
3. afterRender事件主要是图形渲染后再修改或添加图形样式等，需要图形控件支持才可以。
4. Click事件是点击并或者图形相关信息后执行其它动作，最好是图形控件本身能支持，否则比较难处理。

1、源代码示例说明

以D3Chart.js说明如何处理各个事件。

```
refresh : function(container, chartOptions) {
    if(this.svg){
        this.svg.remove();
    }
    this.container = container;
    this._chartOptions = chartOptions;

    this.onBeforeRenderer();//触发事件。

    var margin = {top: 20, right: 20, bottom: 30, left: 100},
        width = chartOptions.chart.width - margin.left - margin.right,
        height = chartOptions.chart.height - margin.top - margin.bottom;

    var x = d3.scale.ordinal()
        .rangeRoundBands([0, width], .1);

    var y = d3.scale.linear()
        .range([height, 0]);
```

在创建图形对象之前，触发beforeRenderer事件，可动态修改图形的高宽等

```
  svg.selectAll(".bar")
      .data(data)
      .enter().append("rect")
        .attr("class", "bar")
        .attr("x", function(d) { return x(d.xValue); })
        .attr("width", x.rangeBand())
        .attr("y", function(d) { return y(d.yValue); })
        .attr("height", function(d) { return height - y(d.yValue); })
        .style("fill", "steelblue");

    this.onAfterRenderer();//触发事件。
```

在图形渲染后，可以拿到图形渲染对象时，触发afterRenderer事件

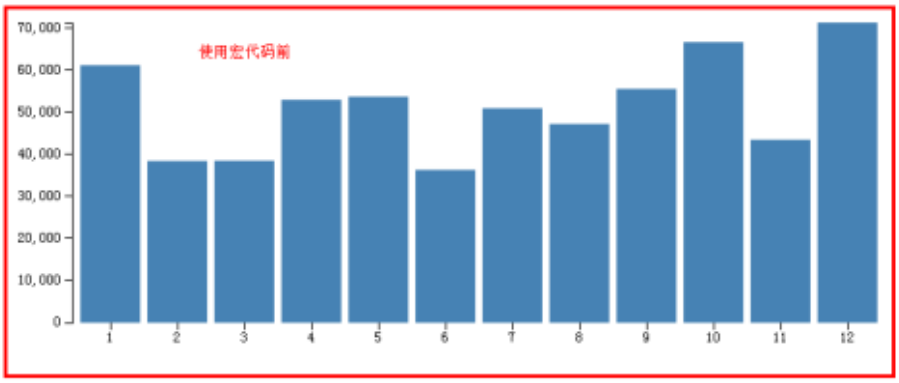
```
    var scope = this;
    svg.selectAll(".bar").on('click',function(pointObj,x,y){
        scope.pointClick(pointObj,x,y);
    })
},
```

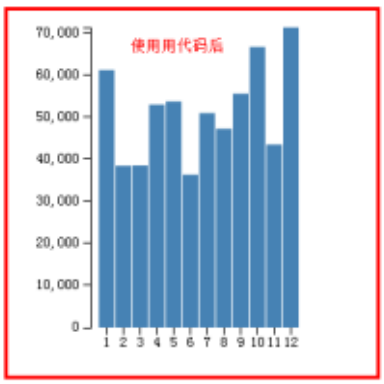
触发点击事件，传递点击对象及坐标信息

2、宏代码示例说明

beforeRenderer事件

```
function main(chartView) {
var option = chartView.getChartObject().getOptions();

option.chart.width = 300;
}
```





五、图形导出实现

目前产品是使用后台模拟浏览器渲染图形，拿到图形生成的svg信息后转成图片导出。
如果新的图形控件不使用svg或不支持获取生成图形的svg信息，会比较麻烦。

六、离线相关

添加相关的js文件到OlapOfflineHandler.java等离线实现文件中，未实现。

原始文档：http://pan.baidu.com/s/1CmZFo