插件开发快速入门

1、搭建开发环境

工欲善其事必先利其器，做开发首先要有个好用的开发工具，并且配置好开发环境。

- 1、如果您是资深开发人员，经常使用Eclipse工具做开发，可以参考 手动配置插件开发环境，搭建好环境后，就可以开始定制功能了。
- 2、如果您是新手，没有Eclipse，可以参考插件开发IDE，直接下载 Smartbi 配置好了的插件开发环境，根据文档做相关设置后，即可以开始定制功能。

文档目录：

2、通过扩展包实现定制功能

2.1、示例说明

下面介绍的 HelloWorld 示例项目源码，可下载HelloDemo.zip 参考。这是一个简单的应用开发，它演示了“如何在资源树的右键菜单添加第三方的功能菜单”的开发过程。具体需求如下：

需求1：通过接口、扩展点插入右键菜单

在Smartbi“定制管理”界面资源树的右键菜单中添加"Hello 163"菜单项，点击"Hello 163"菜单弹出163网站页面。

需求2：前端界面修改

很多时候，客户的需求比特殊，smartbi 并未有相关的接口提供。如果是前端需求，可通过扩展js文件修改。具体需求是：修改定制管理首页的“新建分析”为“新建报表”，并修改背景颜色。

2.2、定制开发过程

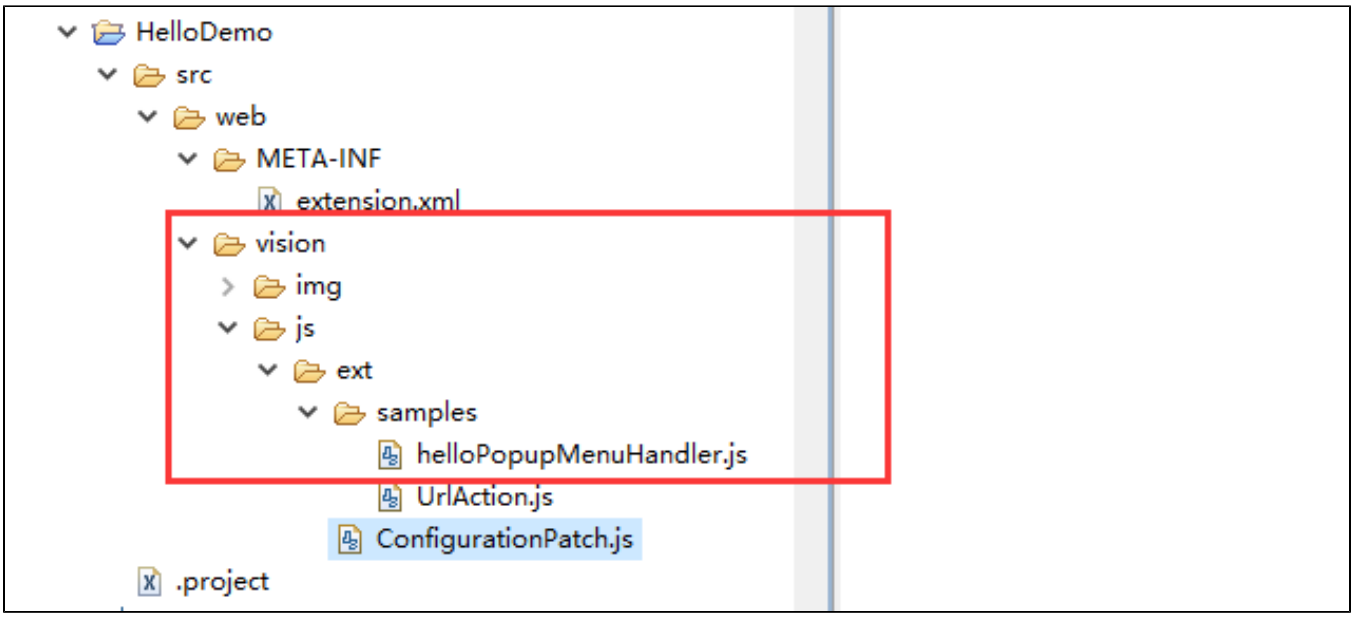
需求1：在"定制管理"界面资源树右键菜单中添加"Hello"菜单项

实现步骤如下：

1、首先添加右键菜单的扩展点。修改扩展包中的 ConfigurationPatch.js 声明文件，添加资源树右键菜单的声明。如下图所示，对 **ConfigurationPatch.js** 声明文件做修改，声明了HelloPopupMenuHandler 这个扩展点。

```
var ConfigurationPatch = {
    extensionPoints : {
        SuperviseTree: {/**定制管理的目录树*/
            handlers: [{
                className : 'ext.samples.HelloPopupMenuHandler'/** 绑定右键菜单事件对象 **/
            }]
        }
    }
}
```

2、根据上图声明，在web目录下创建：vision/js/ext/samples目录，然后创建HelloPopupMenuHandler.js 文件；配置文件中的路径是相对 vision/js 目录的。



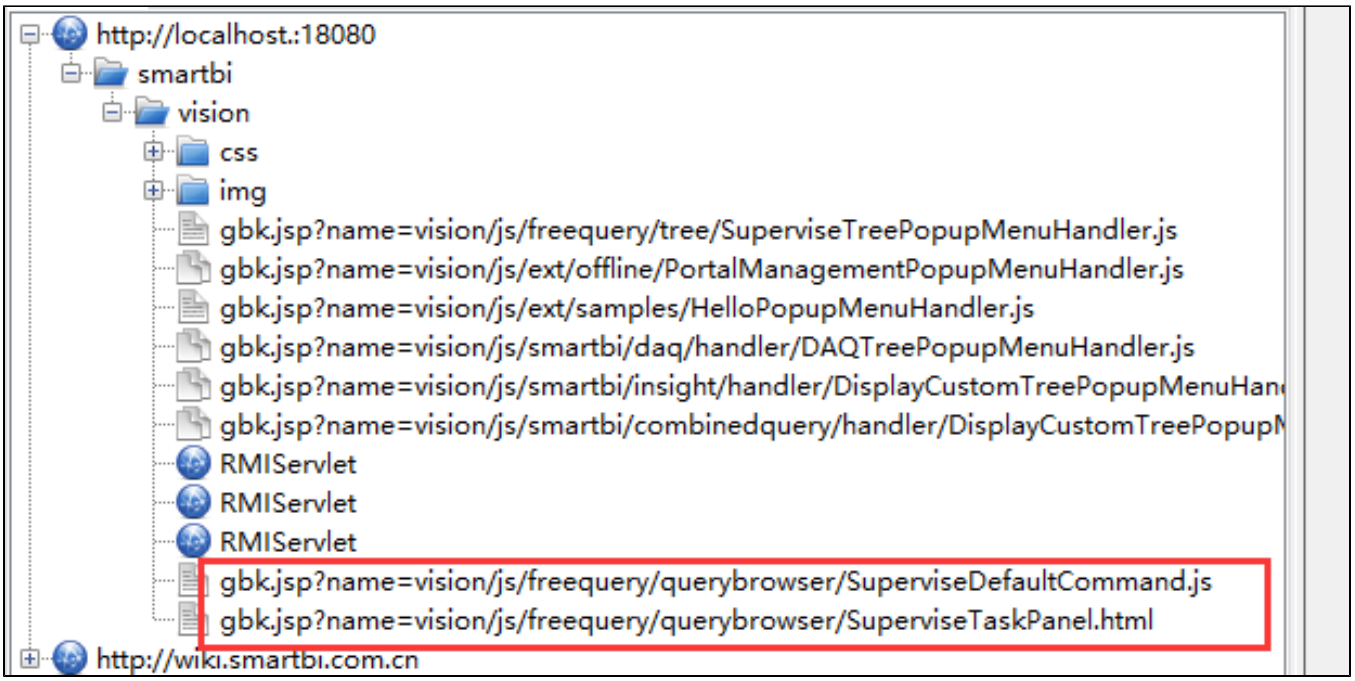
3、实现 HelloPopupMenuHandler.js 文件中的代码，添加新的菜单项。

```
/*
 * 构造函数
 */
var HelloPopupMenuHandler = function(popupMenu) {
    this.popupMenu = popupMenu;
};

/*
 * 析构函数
 */
HelloPopupMenuHandler.prototype.destroy = function() {
}

/*
 * 初始化菜单，只会调用一次。一般在这里创建菜单项
 */
HelloPopupMenuHandler.prototype.initMenu = function() {
    //创建菜单项：参数为【显示名称】、【功能编号（传递到doCmd方法中）】、【快捷键】、【图标】、【插入在哪个菜单项之前】
    this.popupMenu.helloMenu = this.popupMenu.createMenuItem("hello 163", "HelloCommand", "H", null, "刷新");
};

/*
 * 清空菜单状态，每次弹出前调用。一般应在这里隐藏本Handler所创建的所有菜单
 */
HelloPopupMenuHandler.prototype.clearMenuState = function() {
    this.popupMenu.helloMenu.setVisibility(false);
}
```

需求2：修改定制管理首页文字内容及样式

实现步骤：

1、首先找到定制管理首页的实现文件，从war包中找到对应的js文件以及具体的实现方法，在扩展包中修改为需求内容。找到定制管理首页的实现文件（smartbi中很少有页面是使用独立的jsp实现的，基本都是动态拼HTML片段到主界面，定制管理的首页也是如此），可以通过 charles 工具获取（也可以使用 Chrome 浏览器的开发者工具 -> network跟踪）。登录Smartbi 后第一次打开“定制管理”界面前打开 charles，然后点击“定制管理”功能，可以在charles看到下面内容。图片中看到加载了很多js文件，一般情况，最后2个就是我们要找的文件了；也可以通过功能信息猜测相关的文件名称。



2、根据路径找到并打开SuperviseDefaultCommand.js文件。可以看到右边的菜单项是通过数组对象拼装内容的。

```
SuperviseDefaultCommand.prototype.getItems = function() {
    return [ {
        label : "${Newanalysis}",
        bgcolor : "#c1392b",
        columns : 2,
        items : [ {
            id : 'NEW_INSIGHT',
            label : "${Insight}",
            'funcId' : 'CUSTOM_DISPLAYCUSTOM_INSIGHT',
            'licenses' : 'EnterpriseInsight',
            command : [ 'InsightCommand', 'CREATE' ]
        }, {
            id : 'NEW_COMBINED_QUERY',
            label : "${Combinedquery}",
            'funcId' : 'CUSTOM_DISPLAYCUSTOM_COMBINEDQUERY',
            'licenses' : 'WizardQuery',
            command : [ "CombinedQueryCommand", "CREATE" ]
        }, {
            id : 'NEW_DASHBOARD',
            label : "${Instrumentanalysis}",
```

3、在扩展包对应的路径下新建 SuperviseDefaultCommand.js.patch 文件，并修改getItems方法的返回信息。

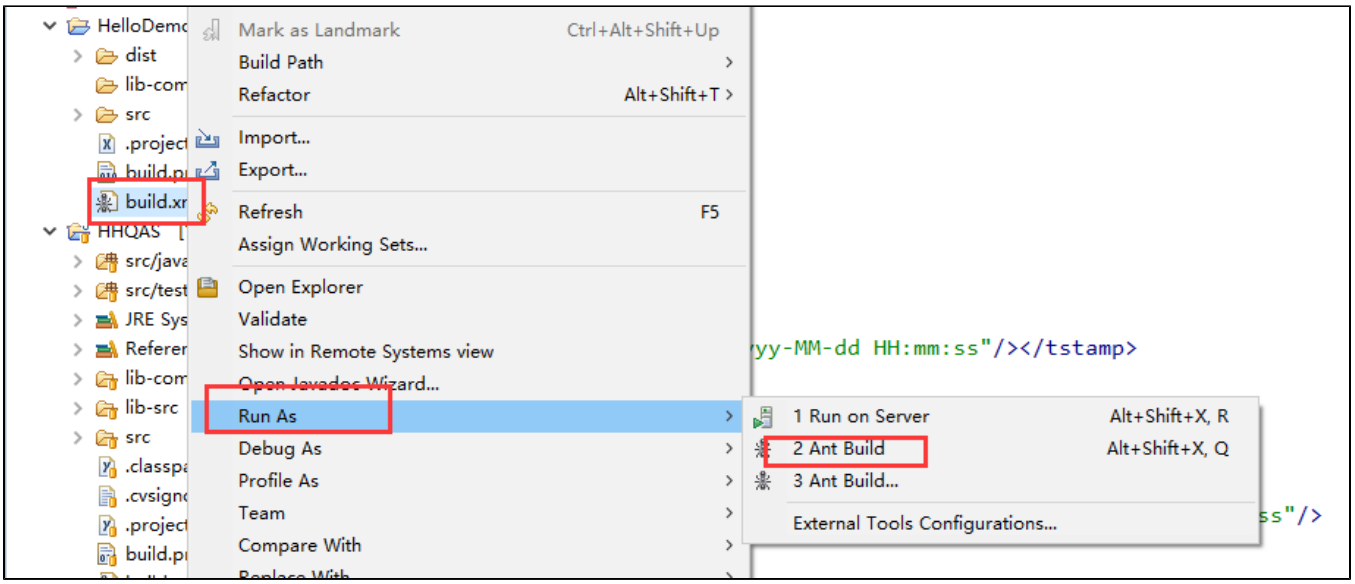
2.3、功能测试和调试

启动应用后即可查看修改效果，如果是客户端的 JS 代码，可以在其中添加 debugger 断点，通过浏览器工具可以单步调试。

```
/** 把源js文件的方法从新命名，以便重用 */
SuperviseDefaultCommand.prototype.getItems_ext = SuperviseDefaultCommand.pr

/**
 * 重新实现旧方法，修改“新建分析”为“新建报表”，并修改背景色
 */
SuperviseDefaultCommand.prototype.getItems = function() {
    debugger;
    var oldItems = this.getItems_ext();
    for(var i in oldItems){
        var items = oldItems[i];
        if(items.label == "新建分析"){
            items.label = "新建报表";
            items.bgcolor = "#000000";
            break;
        }
    }
    return oldItems;
}
```

2.4、打包和部署扩展包

功能开发及测试完成后，直接用 Ant 工具打包，会在 dist 目录下生成 HelloDemo.ext 扩展包，参考“扩展包部署”，把它部署到任意 Smartbi 服务器上。



3、完成HelloWorld程序

至此，就完成了整个 HelloWorld 程序的开发、调试、部署工作。