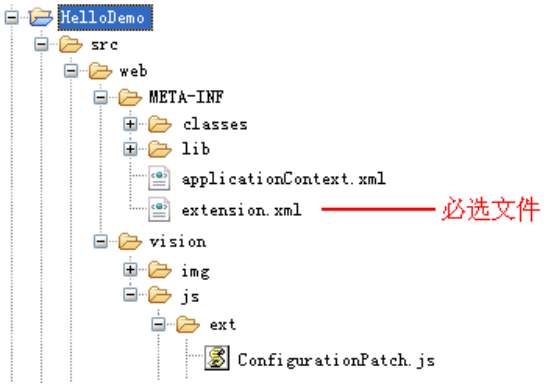
第一课：扩展插件开发基础（内含目录及配置文件介绍）

文档目录：

1. 创建项目工程

使用Eclipse 开发工具，通过"File -> New -> project…"创建一个扩展插件项目工程,也可以使用smartbi提供工具快速创建扩展包，该项目的目录结构如下：



相关目录及文件说明如下：

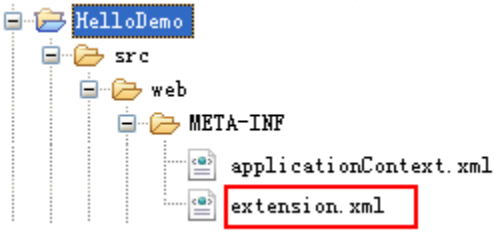
- web目录，是扩展插件的根目录。
- META-INF目录，是扩展插件的配置文件和相关的类存放位置，类似于Java Web项目的WEB-INF目录的作用。
  - classes：包含扩展插件中类文件编译后的class文件（可选）。
  - lib目录：扩展插件引用到的类库。Smartbi已经包括的类库，不应该放在此目录中（可选）。
  - applicationContext.xml：扩展插件Spring配置文件（可选）。
  - extension.xml：扩展插件配置文件（必选）。
- vision目录，是前端文件存放位置。
  - img目录：资源图片存放目录（可选）。
  - js目录：javascript前端文件存放目录（可选）。
  - js\ext\ConfigurationPatch.js：前端配置文件，多个扩展插件会自动合并（可选）。

说明：vision目录中，同路径下的同名文件会替换Smartbi相关文件，因此定制开发的文件一般放到ext目录中。

2. 配置扩展插件

2.1 插件声明文件extension.xml（必要）

extension.xml文件是扩展插件的必要声明文件，存放于META-INF目录中。



基本要素：

属性	描述	可选与否
name	扩展插件的名称。	必选
alias	扩展插件的别名。	必选
desc	扩展插件的描述。	必选

version	扩展插件的版本号。	必选
priority	扩展插件的优先级，默认100。值越小，优先级越高。	可选
before	用于设置扩展包之间的相对优先级，指明当前扩展包应该在某个指定的扩展包之前加载，该属性配置为另一扩展包extension.xml文件的name属性值。	可选
depends	有时多个扩展包之间会有依赖关系，比如扩展包 B 中的 Java 类需要调用扩展包 A 中的方法。 如果直接调用的话，通常会遇到 Caused by: java.lang.NoClassDefFoundError 的错误，提示我们无法找到对应类。 错误的原因是，两个扩展包的 Class Loader 是不一样的。 这时我们就需要设置 B 扩展包的 depends 属性为 A 扩展包了，也是设置为A的extension.xml文件中的name属性值	可选
enable-jsp-processor	支持在扩展插件中使用JSP文件。	可选
file-encoding	声明扩展插件中js、html等文件的编码，默认为UTF-8。	可选
servlet	声明扩展插件中的Servlet对象，请参考web.xml中的结构，见下例。	可选
servlet-mapping	声明Servlet的映射，请参考web.xml中的结构，见下例。	可选
filter	声明扩展插件中的Filter对象，请参考web.xml中的结构，见下例。	可选
filter-mapping	声明需要过滤的url映射，请参考web.xml中的结构，见下例。	可选

extension.xml文件内容示例

```
<?xml version="1.0" encoding="GBK"?>
<extension name="KingbaseSmartbiExtension" alias="KingbaseSmartbiExtension" desc=" SmartbiExtension Samples"
version="1.0">
     <enable-jsp-processor/>
     <file-encoding>GBK</file-encoding>
        <!--servletservlet-mapping-->
        <servlet>
                <servlet-name> TestServlet </servlet-name>
                <servlet-class>bof.extension.test.TestServlet</servlet-class>
                <init-param>
                        <param-name>x</param-name>
                        <param-value>xv</param-value>
                </init-param>
                <init-param>
                        <param-name>y</param-name>
                        <param-value>yv</param-value>
                </init-param>
                <!--servlet-->
                <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name> TestServlet </servlet-name>
                <url-pattern>/TestServlet</url-pattern>
        </servlet-mapping>
        <!--filterfilter-mapping-->
    <filter>
                <filter-name>TestFilter</filter-name>
                <filter-class>bof.extension.test.TestFilter</filter-class>
        </filter>
        <filter-mapping>
                <filter-name>TestFilter</filter-name>
                <url-pattern>/vision/ssreportServlet</url-pattern>
        </filter-mapping>
</extension>
```

2.2 Spring声明文件applicationContext.xml（可选）

applicationContext.xml文件是扩展插件中的Spring声明文件，存放于META-INF目录中。

该文件是可选的。在java代码中调用服务器端SDK提供的接口时不需要本文件；

仅当需要在扩展插件中使用Smartbi内部模块的方法，或将新增模块组件注册到Framework和RMIModule中时（譬如需要在扩展包实现服务端方法供前端使用），才需要定义此文件，详细示例见自定义模块组件（Module）。

应用说明：

- 当需要在扩展包实现服务端方法供前端调用时使用。
- Smartbi使用Spring加载服务器上的组件，因此当扩展插件需要使用到Smartbi原有功能时必须新增组件也使用Spring进行组件加载。
- 在Smartbi启动过程中会自动加载smartbi.war\WEB-INF\applicationContext.xml初始化内置组件，然后再按扩展插件顺序加载扩展插件中的applicationContext.xml。
- 在扩展插件中的applicationContext.xml可以直接引用Smartbi内置的对象，只需要通过相应的id引用就可以，例如下面示例，当然需要在module中声明对应组件的get和set方法。

Smartbi常用内置的对象说明如下（更多内置对象请见smartbi.war\WEB-INF\applicationContext.xml）：

ID	类型	描述
dao	接口：smartbi.repository.IDAOModule 实现类：smartbi.repository.DAOModule	处理数据库相关操作。
state	接口：smartbi.state.IStateModule 实现类：smartbi.state.StateModule	处理会话操作。
catalogtree	接口：smartbi.catalogtree.ICatalogTreeModule 实现类：smartbi.catalogtree.CatalogTreeModule	处理资源树操作。
usermanager	接口：smartbi.usermanager.IUserManagerModule 实现类：smartbi.usermanager.UserManagerModule	处理用户管理操作。
rmi	实现类：smartbi.framework.rmi.RMIModule	声明前端请求对应的模块。
framework	接口：smartbi.framework.IFramework 实现类：smartbi.framework.Framework	处理模块的升级和激活POJO对象。

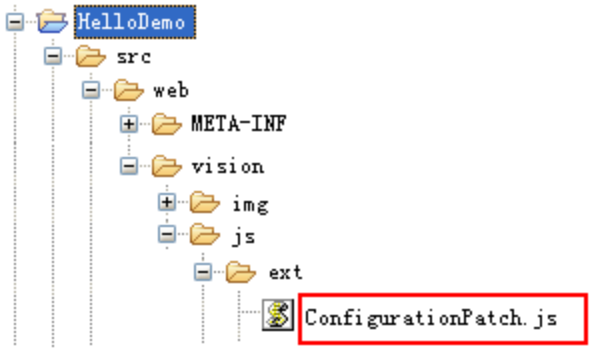
- 下面示例重新声明内置对象framework和rmi，用于注册新的组件testExt，这种声明方式并不会完全覆盖产品中的framework和rmi，而是会自动合并到内置对象中的List和Map属性。

applicationContext.xml文件内容示例

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans>
	<bean id="framework" class="bof.framework.Framework" factory-method="getInstance">
		<property name="modules">
			<map>
				<entry>
					<key>
						<value>TestExt</value>
					</key>
					<ref bean="testExt" />
				</entry>
			</map>
		</property>
	</bean>
	<bean id="rmi" class="bof.framework.rmi.RMIModule" factory-method="getInstance">
		<property name="modules">
			<map>
				<entry>
					<key>
						<value>TestExt</value>
					</key>
					<ref bean="testExt" />
				</entry>
			</map>
		</property>
	</bean>
	<bean id="testExt" class="bof.extension.test.TestExtensionModule" factory-method="getInstance" >
		<!-- daoTestExtensionModuledaoModulegetset -->
		<property name="daoModule">
			<ref bean="dao" />
		</property>
	</bean>
</beans>
```

2.3 扩展点配置文件ConfigurationPatch.js（可选）

在产品中存在一个js配置文件Configuration.js(目录smartbi.war\vision\js\中，了解这个文件大概也能了解产品前端功能框架)，扩展包中存在一个可选的扩展配置文件ConfigurationPatch.js（存放于扩展插件中的vision\js\ext目录中）实际就是对产品Configuration.js文件的扩展，通过配置相应扩展点达到实现某个功能的目的，譬如新增系统选项。



该文件用于配置smartbi提供的修改产品内置功能的JS扩展点，**ConfigurationPatch.js**包含两个属性，分别是extensionPoints、patches的属性：

- extensionPoints是客户端扩展点内容的定义，客户端会根据扩展点内容的声明作出相应的处理，Smartbi会自动合并原有Configuration.js和所有扩展插件中的声明。extensionPoints对象支持以下常用属性：

属性	用途
MainFrame	可以在MainFrame.actions下添加一级菜单模块。
Custom、Manager等	在相应一级模块下添加二级模块、任务面板操作入口。

CatalogTree	添加右键菜单。

- Patches提供了更丰富的属性定义，并且可以对extensionPoints中原有定义的属性进行修改。比如修改Smartbi原有扩展点内容，或在指定位置插入新的扩展点。每个Patch对象包含以下属性：

属性	描述
path	表示该path要修改的Configuration.js中的对象路径。
key	需要替换的键值，可以为数字或者字符串，当operation为appendObject时忽略此项。 • 0和正数：比如0，代表从左到右第一个键值 • 负数：比如-1，代表从右到左第一个键值
operation	path的动作，共有以下几种： • replace：替换原来的值。 • append：添加到原有数组中，value可以为数组或对象。 • appendObject：添加到原有对象中，value为是对象。 • insert：插入原有数组中。 • remove：删除原有对象中的值。
value	修改后的值，当operation为remove时忽略此项

- extensionPoints、Patches均为数组，可以定义多个扩展点对象。

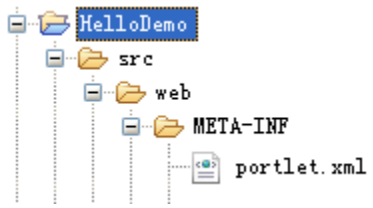
说明
Smartbi可供扩展的所有扩展点请参阅"系统扩展点"。

扩展点声明示例

```
var ConfigurationPatch = {
        /**/
patches: [
                {
                        path: "/extensionPoints/Custom/actions",
                        key: 6,
                        operation: "remove"
                }
        ],
        extensionPoints : {
                /* */
MainFrame : {
                        actions : [
                                {
                                        className :"ext.b.FibonacciAction",
                                        groupId :"default"
                                },
                                {
                                        className :"ext.b.NumberSeriesAction",
                                        groupId :"default"
                                }
                        ]
                },
        /* */
                Custom: {
                        actions: [
                                {
                                        className :"ext.b.FibonacciAction",
                                        groupId :"default"
                                }
                        ],
             /* */
                        DatasetTaskPanel : {
                                config: {
                                        handlers:[
                                                {className:"ext.Custom.CustomTaskPanel"}
                                        ]
                                }
                        }
                },
         /*""->"" */
                CatalogTree: {
        displayCustomHandler:[{className : "ext.b.DisplayCustom_TreePopupMenuHandler"}]
                }
        }
};
```

2.4 自定义Portlet声明文件portlet.xml（可选）

smartbi有一种资源叫“页面”，Portlet就是可以拖到页面的组件（用户使用时，新建页面>添加资源 页面左侧会有portlet资源，譬如产品带有IPAD分页部件），portlet.xml文件是扩展插件中的自定义portlet声明文件，存放于META-INF目录中，请见自定义Portlet。



Smartbi本身提供web链接、灵活分析、仪表分析、地图分析、多维分析、复杂报表、指标报表、目录等多种portlet资源。自定义 portlet用于实现产品未支持的资源类型，它必须满足Portlet和PortletEdit的接口规范。

自定义Portlet基本要素：

属性	描述
Title	portlet的默认标题。
Icon	portlet在资源树节点上显示的图标,图标文件放在src/web/img/catalogtree目录下。
portlet-name	portlet名称，portlet的标识。
display-name	portlet显示名称，添加portlet时界面上显示的名称。
description	portlet描述信息。
is-bof-tree-node	是否是资源树上的节点对象。
portlet-viewer-js-class	展示界面对应的文件。
portlet-editor-js-class	编辑界面对应的文件。

文件内容示例

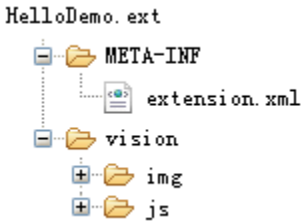
```
<?xml version="1.0" encoding="UTF-8" ?>
<portlet-app>
        <portlet title="HelloWorld" icon="HelloWorld.gif">
        <portlet-name>HelloWorld</portlet-name>
        <display-name>HelloWorld</display-name>
        <description>HelloWorld</description>
                   <is-bof-tree-node>false </is-bof-tree-node>
                <portlet-viewer-js-class>csdc.HelloWorldPortlet</portlet-viewer-js-class>
                <portlet-editor-js-class>csdc.HelloWorldPortletEdit</portlet-editor-js-class>
        </portlet>
 </portlet-app>
```

3. 打包扩展插件

如果是使用Eclipse开发，可以直接右键扩展包中build.xml文件>Run As > Ant Build打包，默认会打包到扩展包开发目录\dist目录下。

也可以使用jar命令将扩展插件打包，以ext为扩展名。操作步骤如下：

1. 启动命令行提示符cmd；
2. 通过cd命令进入到项目所在的src\web目录，例如：D:\HelloDemo\src\web\；
3. 输入命令：c:\ jdk1.5.0_01\bin\jar.exe cvf HelloDemo.ext .
4. 回车后，即将HelloDemo\src\web目录下的所有内容打包进HelloDemo.ext文件。



4. 加载扩展插件

4.1 通过配置界面对扩展插件进行加载

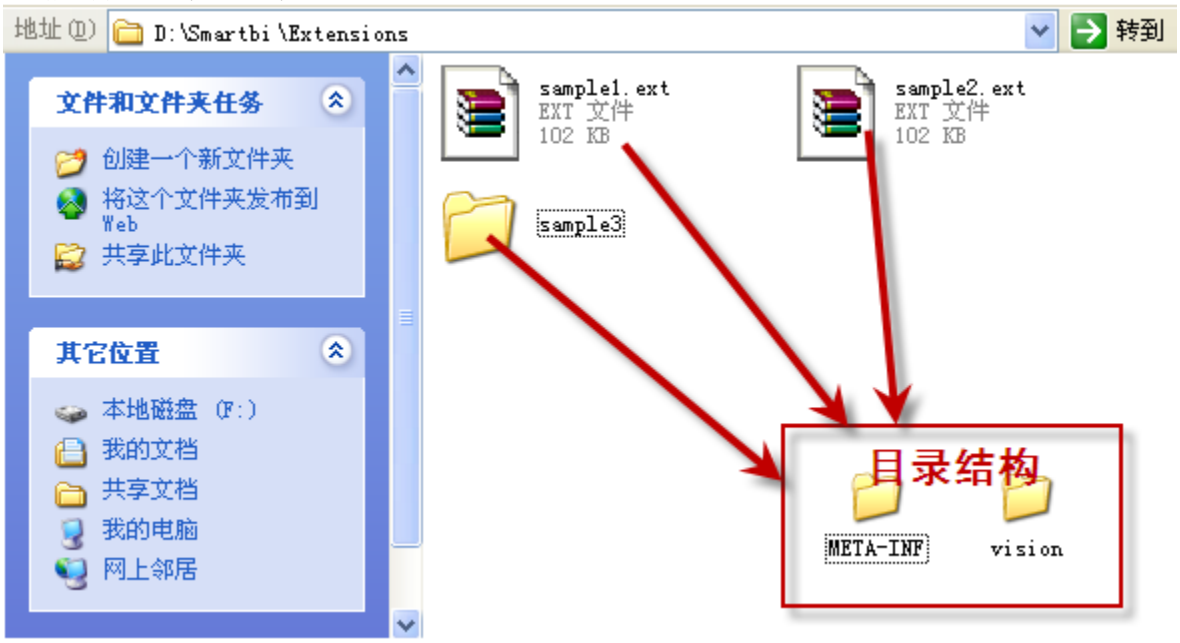
加载单个插件：

1. 启动Smartbi应用服务。

2. 访问http://ip:host/smartbi/config页面。

3. 设置扩展插件存放路径。如：D:\ Smartbi\Extensions。
4. 保存设置。
5. 重新启动Smartbi应用服务器。

加载多个插件：
为了能够同时加载多个扩展插件，把sample1.ext、sample2.ext和sample3三个扩展插件存放在D:\ Smartbi\Extensions目录中。在config页面，设置扩展包存放路径为：D:\ Smartbi\Extensions。

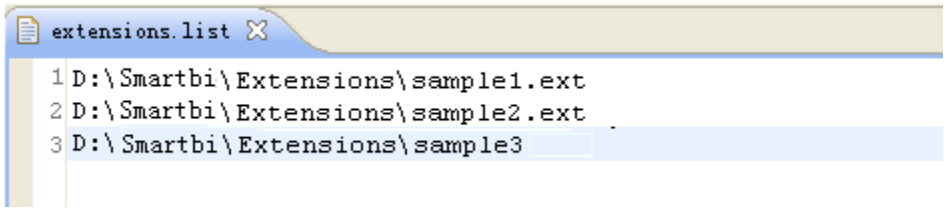


说明：

- 一般情况下建议将扩展插件打包为*.ext文件，但系统也支持直接将开发的扩展插件src\web目录直接拷贝到扩展包的加载目录，这样该扩展插件也能被加载。如上图所示，sample1和sample2分别打包为 ext 文件了，而sample3则直接拷贝的src\web目录，这三个扩展包都会被正确加载。
- 扩展插件加载顺序参看常见问题：扩展插件的加载策略是什么。

4.2 通过修改extensions.list文件对扩展插件进行加载

通过修改smartbi.war/WEB-INF/extensions/extensions.list文件，对扩展插件进行加载。文件中的每一行为一个扩展插件路径的声明。如图所示：



说明：

- 扩展插件加载顺序参看常见问题：扩展插件的加载策略是什么。

4.3 列出已加载扩展插件

登录Smartbi，通过**【定制管理】-【系统运维】-【调试工具集】-【列出已加载扩展插件】**，查看所部署的扩展插件是否已正确加载。

名称	别名	描述	版本	加载路径
SmartbiExtension	SmartbiExtension	SmartbiExtension Samples	1.0	E:/Tools/eclipse/workspace/SmartbiExtension/src/web