HUNDSUN®

恒生统一客户端开发平台 V1.1

开发指南

恒生电子股份有限公司 公司研发中心

文档版本 01

发布日期 2013-12-31

恒生电子股份有限公司 公司研发中心

地址： 杭州市滨江区江南大道 3588 号恒生大厦 14 楼 邮编：310053

网址： http://rdc.hundsun.com

客户服务电话： 0571-28829563

客户服务传真： 0571-28823456

客户服务邮箱： service.rd@hundsun.com

商标声明

HUNDSUN 天堂恒生® 你我他® 为恒生电子股份有限公司的注册商标。

注意

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目录

前言

目的

本文是关于恒生统一客户端开发平台(HSRCF，Hundsun Reused Client Framework)的开发指南，它将阐述如何使用 HSRCF 进行 Windows 桌面客户端快速开发的各方面知识。

预期读者

使用 HSRCF 进行业务模块开发的人员。

手册概况

本手册各章节内容如下表所示。

章节	内容
1 HSRCF 概述	本章主要介绍 HSRCF 的定义、目标、特性和主界面。
2 开发前准备	本章主要介绍使用 HSRCF 开发之前需要的准备工作。
3 开发 Hello HSRCF 插件	本章主要以开发 Hello HSRCF 插件为例，介绍开发插件的步骤。
4 插件生命周期	本章主要介绍插件的生命周期和主要处理接口。
5 工作区	本章主要介绍工作区的使用方法。
6 自定义框架页	本章主要介绍如何自定义框架页。

章节	内容
7 客户端通信	本章主要介绍如何进行客户端通信开发。
8 客户端身份标示	本章主要介绍 HSRCF 为业务开发提供的客户端身份标示接口。
9 日志和异常	本章主要介绍 HSRCF 提供的日志和异常接口。
10 控件	本章主要介绍 HSRCF 实现的控件。
11 界面布局	本章主要介绍如何进行界面布局。
12 界面数据校验	本章主要介绍界面数据的校验方法。
13 界面按键控制	本章主要介绍界面的按键控制方法。
14 多语言	本章主要介绍多语言的使用方法。
15 常用接口和对象	本章主要介绍业务开发过程常用的接口和对象。
附录 A Visual Studio 安装	本章主要介绍 Visual Studio 方法。
附录 B C#语言和.NET Framework 介绍	本章主要介绍什么是 C#语言和.NET Framework。
附录 C Windows Form 介绍	本章主要介绍什么是 Windows Form。

缩略语/术语

下面列出了本手册中出现的缩略语和术语。

H

HSRCF	Hundsun Reused Client Framework	恒生统一客户端开发平台

M

MC2.0	Message Center	消息中心

T

T2	T2 协议	恒生金融基础件 2.0 所支持的基于 TCP/SPX 的增强型通信应用协议，支持长功能号、基于许可证的接入控制；与 T1 协议不兼容，被规划用作 T1 协议的替代协议对外开放。

文档版本

产品版本	文档版本	更新内容
HSRCF1.1	01（2013-12-31）	全新手册

1 HSRCF 概述

1.1 HSRCF 是什么

HSRCF(Hundsun Reused Client Framework,恒生统一客户端开发平台)是一个稳定、安全、高效的.NET 智能客户端二次开发平台，也是便捷、易用的快速业务开发框架，开发语言为 C#。

1.2 HSRCF 目标

目前，公司大部分客户端都采用 Delphi 语言开发实现，但随着 Delphi 语言的日益趋弱，公司希望能有一个新的客户端框架平台来慢慢过渡并丰富现有的产品线。因此经过各方论证，公司决定使用.NET 平台重新设计一套稳定、安全、高效、复用性强的客户端二次开发平台。

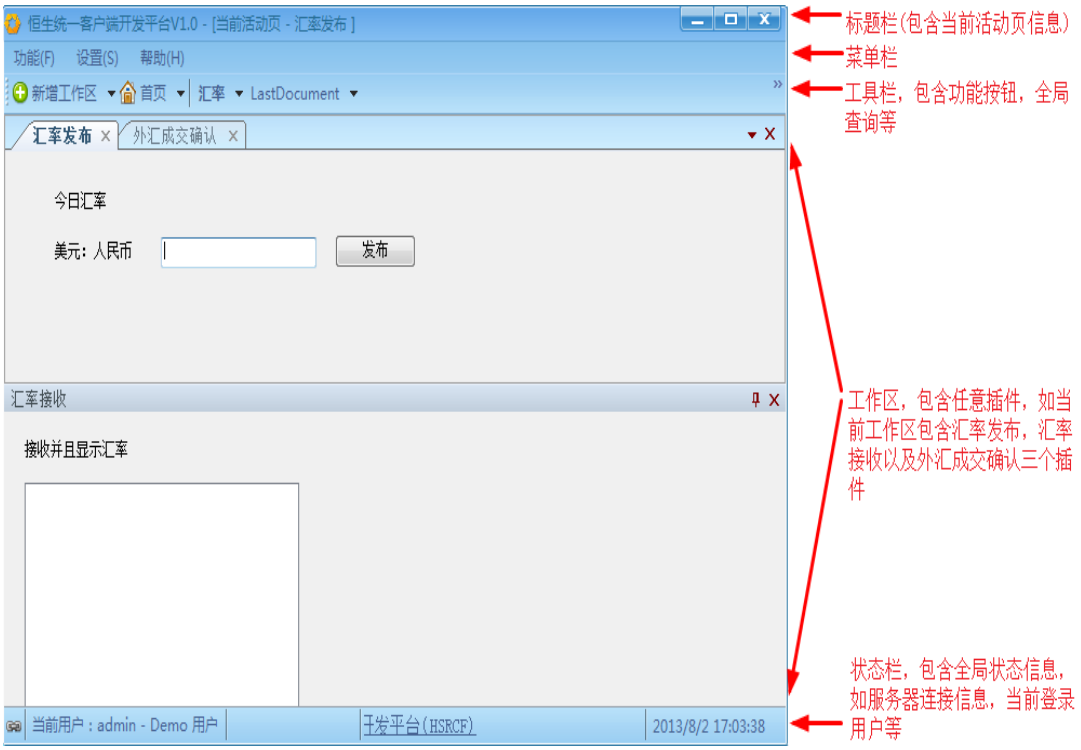
1.3 HSRCF 特性

HSRCF 有很多特性，例如插件式开发、工作区管理、窗体联动、自定义框架页、布局、校验、换肤、多语言、依赖注入、权限、缓存、日志、控件库和通信库等。下表对各个特性进行了简要描述。

特性名称	描述
插件式开发	提供插件开发管理，包括安装、卸载等。
工作区管理	可以自定义工作区。
自定义框架页	可以自定义工具栏、状态栏、菜单栏。
通信库	支持 T2、MC 协议等。

特性名称	描述
控件库	除了支持标签的系列控件外，还编写了带标签的输入性控件，可以称其为**带标签组合控件。**
换肤	通过设置整体框架的主题，可以支持替换当前的皮肤。
日志	提供日志类型、日志级别。
异常	框架对未捕获的异常将进行日志记录并提示，其他的业务插件异常由业务各自处理。
界面校验	支持控件内置属性校验和编写自定义校验事件，以及按钮的自动校验事件等。
界面布局	支持网格控件的列布局，支持非嵌套容器内的输入性控件的动态表格形式布局。
界面上下键控制	目前对插件内控件的上下键进行全局控制。
客户端身份标示	客户端的身份标示，如操作员或登录用户，在某一个客户端中是全局唯一的。
国际化	通过资源文件提供多语言支持。
依赖注入	通过 Dependency 等特性注入。
常用工具类	常用的工具类。

1.4 HSRCF 界面



在了解 HSRCF 后，就可以进行开发过程了。

2 开发前准备

在使用 HSRCF 开发前需要一些准备工作，例如搭建开发环境、连接开发目录和运行时目录等。

2.1 开发环境搭建

目前建议的开发环境如下表所示。

操作系统	Windows 7/XP
开发工具	Visual Studio 2010
.NET Framework 版本	4.0
版本管理客户端	TortoiseSVN (SVN 客户端)和AnkhSVN (Visual Studio SVN 插件)

注意

如果系统是 XP 系统，保证系统已经安装 VC++库，例如可以选择安装如下的分发包：Microsoft Visual C++ 2005 SP1 Redistributable Package (x86)

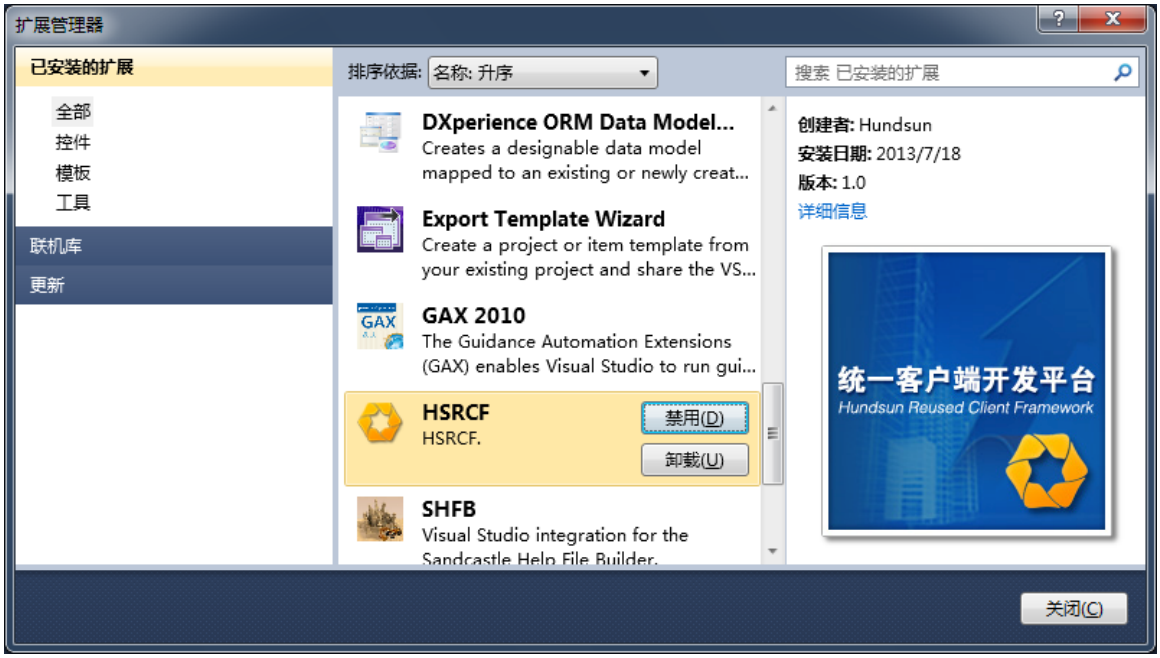
2.1.1 Visual Studio 安装

关于 Visual Studio 的安装过程，请查看附录 A Visual Stuio 安装。

2.1.2 Visual Studio 扩展包安装

平台提供了 Visual Studio 的扩展包，它能够在新建插件项目和新建插件时提供模板支持。

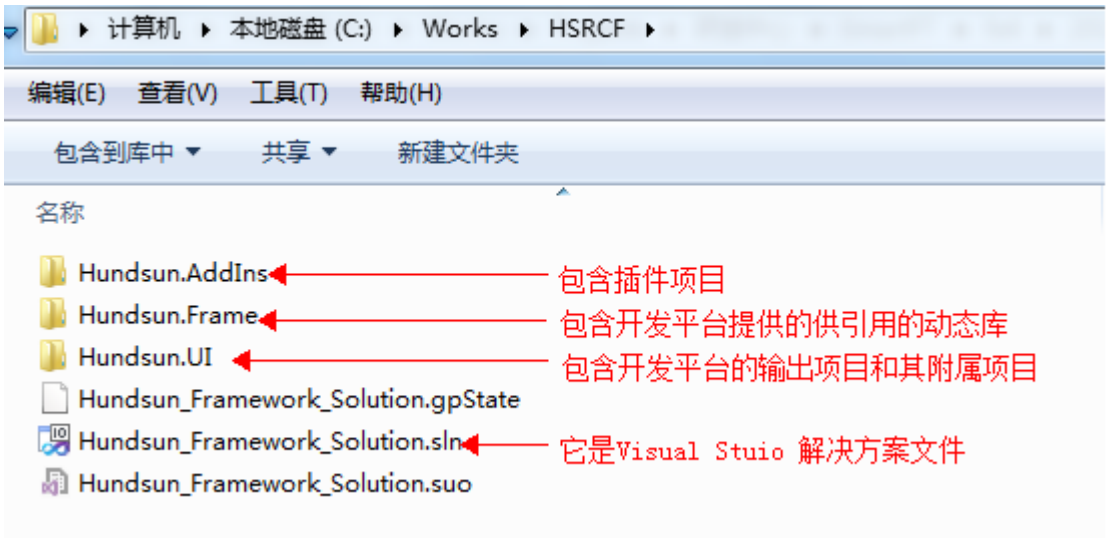
直接双击 图标进行安装，安装完后重新打开 Visual Studio，点击“菜单工具 > 扩展管理器”，将看到扩展管理器界面，选择 HSRCF 扩展包，用户可对其进行禁用/启动、卸载操作。



如果启用了 HSRCF 扩展包，可以在添加 Visual Studio 项目后选择 HSRCF 提供的 HSRCF 插件项目模板、HSRCF 插件模板、HSRCF 窗体模板以及 HSRCF 控制器类模板。

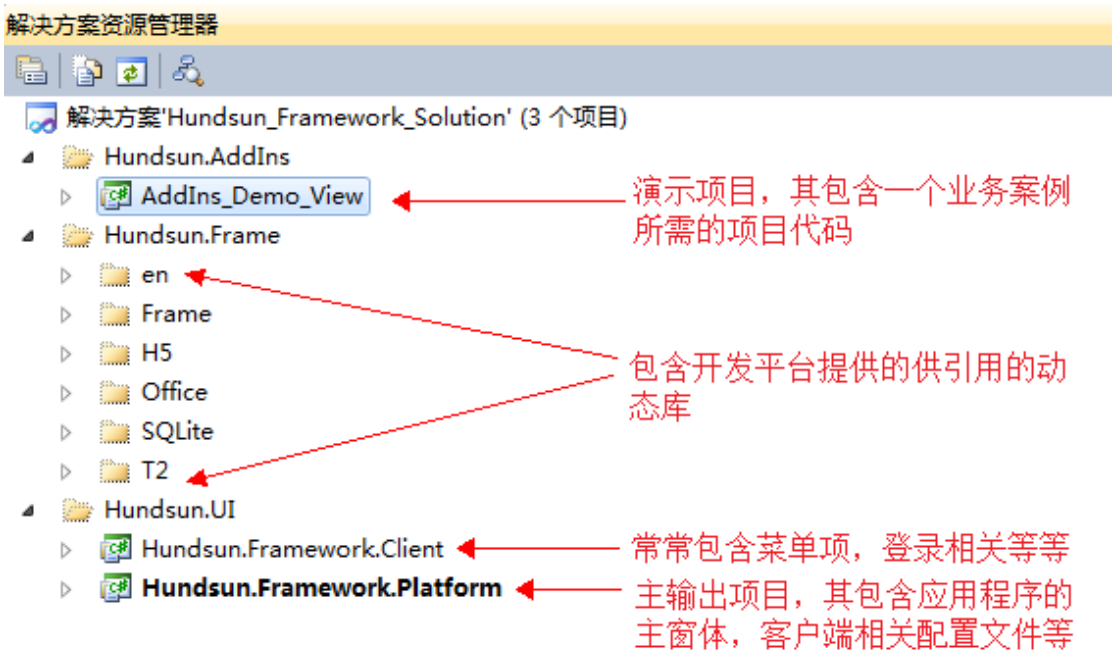
2.2 源码目录结构

平台提供了解决方案压缩包 HSRCF.zip，请解压 HSRCF.zip 到指定目录，假设解压到 C:\Works\HSRCF，将看到如下图所示界面，该界面呈现了开发平台的源码存放的目录结构。



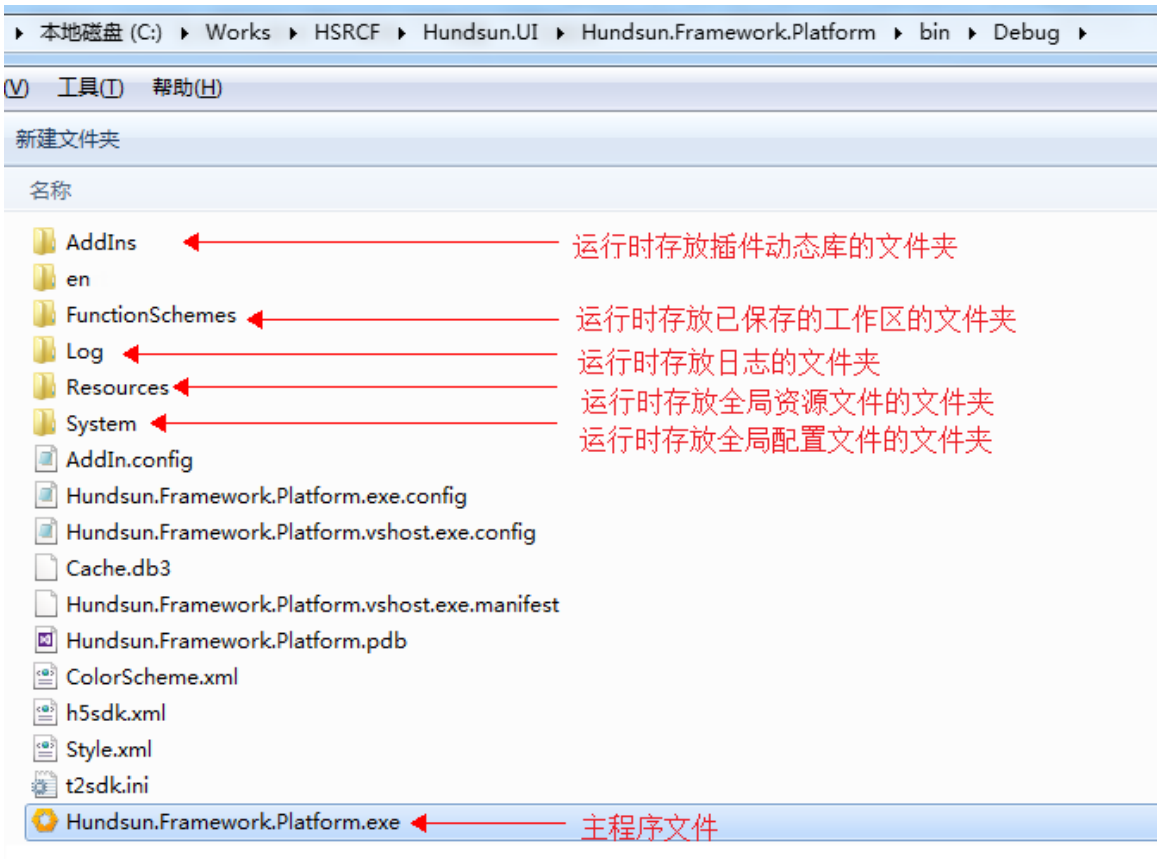
2.3 解决方案项目结构

双击 Hundsun_Framework_Solution.sln，它将自动打开 Visual Studio 开发环境，在视图菜单中打开解决方案管理器，可以看到如下的界面。



2.4 运行时执行目录结构

编译解决方案后，在文件系统中打开 Hundsun.Framework.Platform 项目下的输出目录(Debug 情况默认输出目录为项目文件夹下的 bin\Debug\)，该目录即开发平台的运行时执行目录，目录如下图所示。



2.5 引用类库说明

框架提供的类库如下表所示。这些类库都存放在源码目录下的 Hundsun.Frame 子目录下。

名称	描述
Hundsun.Framework.AddIn.dll	提供统一的插件管理机制，建立插件联动模式，通过发布订阅事件系统及共享 State 实现了对象间的通讯。
Hundsun.Framework.AddIn.WinForms.dll	提供综合屏展示功能，便于应用程序的界面扩展。

名称	描述
Hundsun.Framework.IoC.dll	提供一个 IOC 界面容器的接口规范，并带有通过策略和配置信息自动创建对象实例的对象构造器。
Hundsun.Framework.MVP.dll	继承 Hundsun.Framework.IoC 的接口，提供应用程序的组成部件，其中包含 SmartParts，支持 Service、业务逻辑和配置信息等。
Hundsun.Framework.MVP.WinForms.dll	通过继承自 FormShellApplication 的应用程序类来启动。FormShellApplication 需要传入两个类型参数，分别继承自 WorkItem 的类和继承自 Form 的类（应用程序主界面 FormShell），主界面上可以放置供所有界面视图使用的公共的 UI Element (如 Toolbar)和显示用户界面的 WorkSpace。 WorkItem 是封装了用例实现的容器，容器中的对象可以共享信息。WorkItem 也可以包含下级 WorkItem。
Hundsun.Framework.Communication.dll	该动态库封装了 T2、MC 等协议，详细信息可以查看客户端通信章节。
Hundsun.Framework.HSControls.dll	该动态库封装了恒生体系的控件库。它除了实现标准控件之外，还实现了带标签系列的控件、多列下拉控件以及可折叠的面板等。详细可以查看控件章节。
Hundsun.Framework.UIFrame.dll	该动态库对界面框架（如常用基类，表格控件列布局，按键控制等）的封装。
Hundsun.Framework.Entity.dll	该动态库封装了常用的工具类。详细可以查看常用工具类章节。
Hundsun.Framework.Tools.dll	该动态库封装了常用的工具栏和状态栏等工具项。

3 开发 Hello HSRCF 插件

3.1 概述

3.1.1 插件是什么

插件(Plug-in,又称 addin、add-in、addon 或 add-on,又译外挂)是一种遵循一定规范的应用程序接口编写出来的程序。HSRCF 插件也是同样的原则，HSRCF 插件需要继承指定的类并标注相关特性，即可视为插件，它需要生存于 HSRCF 的插件容器中。HSRCF 插件必须继承 AddInPartBase，并标注为 AddInPart 特性，详细请参见基类继承和特性约束章节。

3.1.2 HSRCF 插件优点

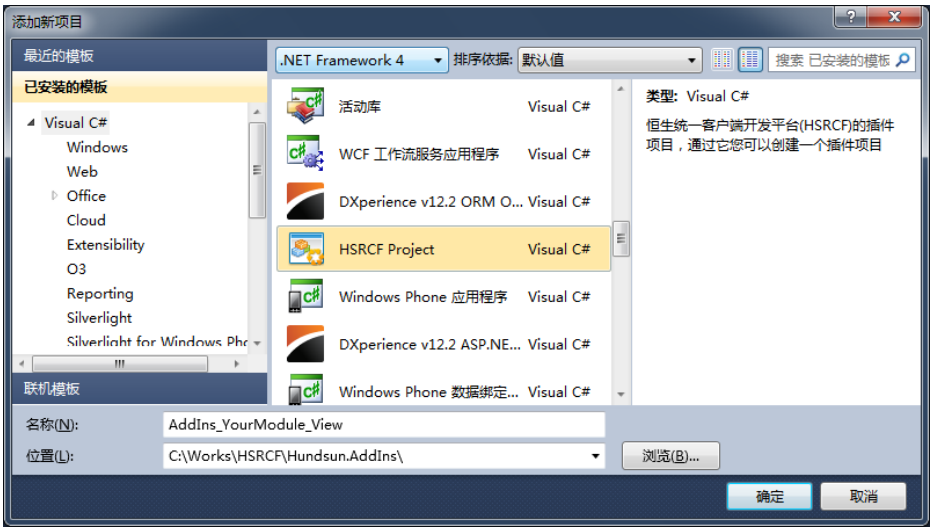
使用 HSRCF 进行插件式开发，有以下优点：

- 各业务单元相对独立的开发、部署、维护
- 业务插件之间不相互影响
- 可根据需求动态的组装、分离业务系统

3.2 新建插件项目

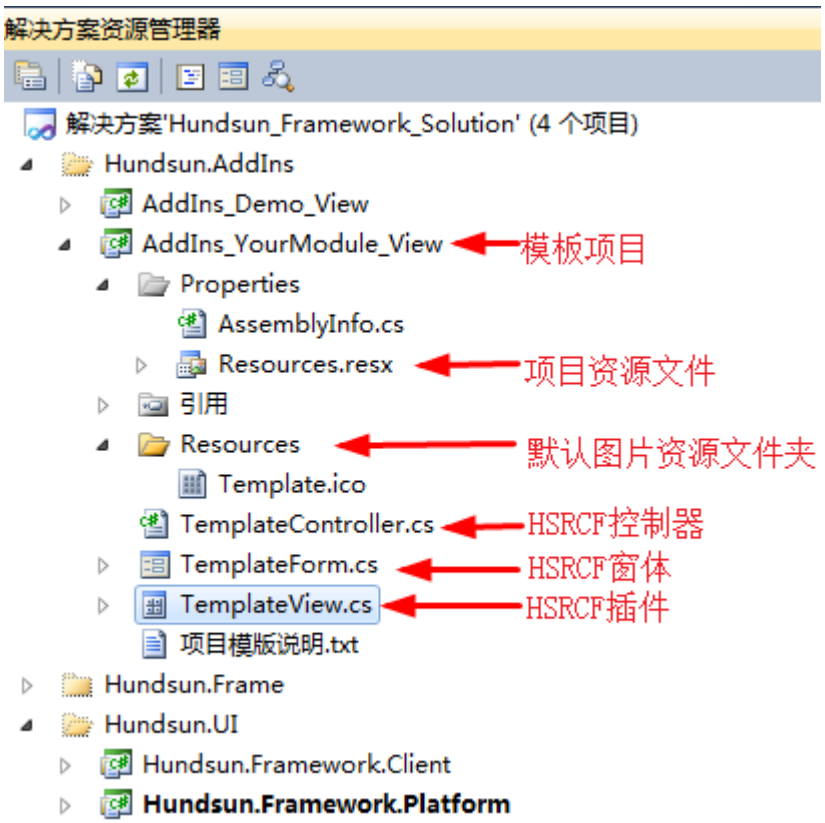
新建插件项目很简单，因为平台已经提供了 Visual Studio 项目模板来创建插件项目。以下是新建插件项目的步骤：

1.) 如果启用了HSRCF 扩展包，在解决方案项目结构一节，可以看到 Hundsun.AddIns 虚拟文件夹，右键单击该节点，选择添加一个新建项目，将打开**添加新项目** 窗体，如下图所示。



2.) 可以选择 **HSRCF Project** 新建 **HSRCF 插件项目**，输入插件项目名称，并选择插件项目的文件系统位置（**该位置必须指定为 Hundsun.AddIns 文件夹下**，如 C:\Works\HSRCF\Hundsun.AddIns\，可以参考源码目录结构章节了解整个开发平台的源码存放的目录结构），然后单击“确定”。

通过上面的步骤，就可以创建一个插件项目，如下图所示。

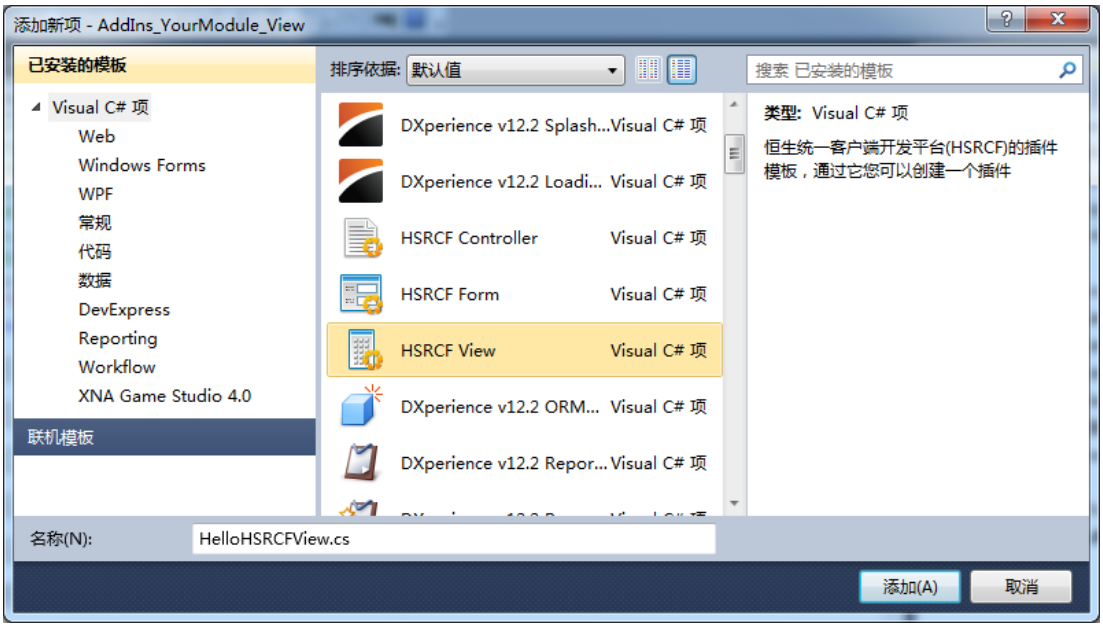


可以直接使用该项目模板生成的 HSRCF 插件，也可以重新添加插件，下文会介绍如何通过插件模板添加插件。

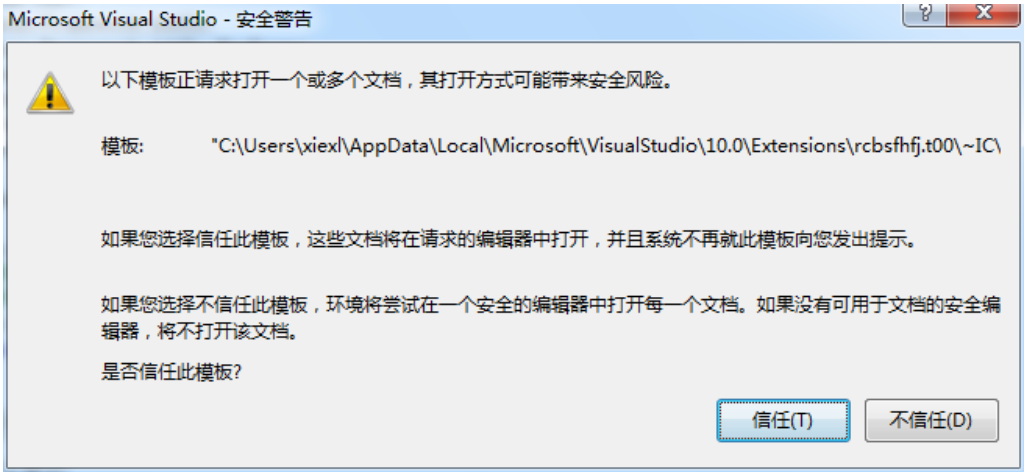
3.3 新建插件

新建插件同样也很简单，因为已经提供了 Visual Studio 插件项模板创建插件。以下是新建插件的步骤：

1.) 如果启用了HSRCF 扩展包，请选择一个插件项目，右键单击选择“添加新建项”；左侧模板分类树中选择 Visual C#，右侧模板区域中选择 **HSRCF View** 模板，如下图所示。



2.) 输入一个插件名称(如 HelloHSRCFView)，单击“添加”按钮，此时可能会弹出一个安全警告，如下所示。

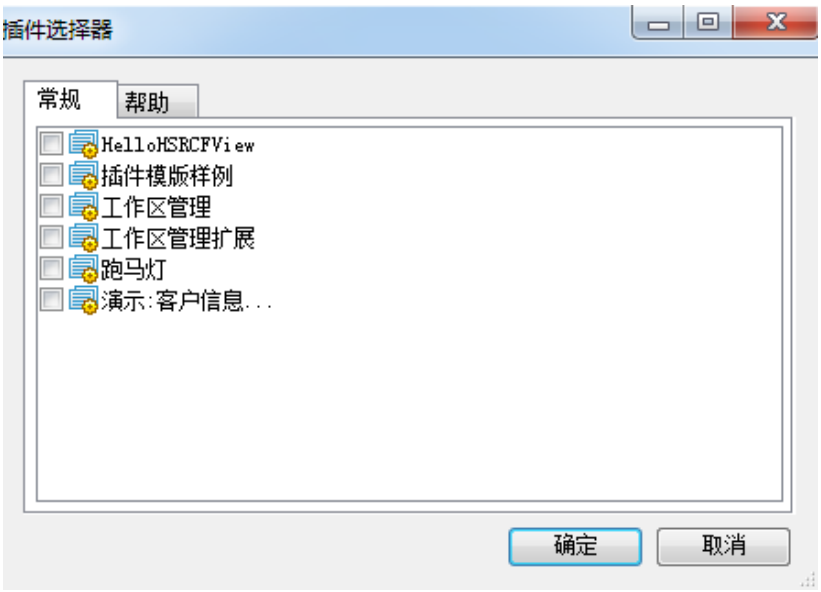


3.) 请选择信任此模板，这样就添加了一个 HSRCF 插件，如下所示。

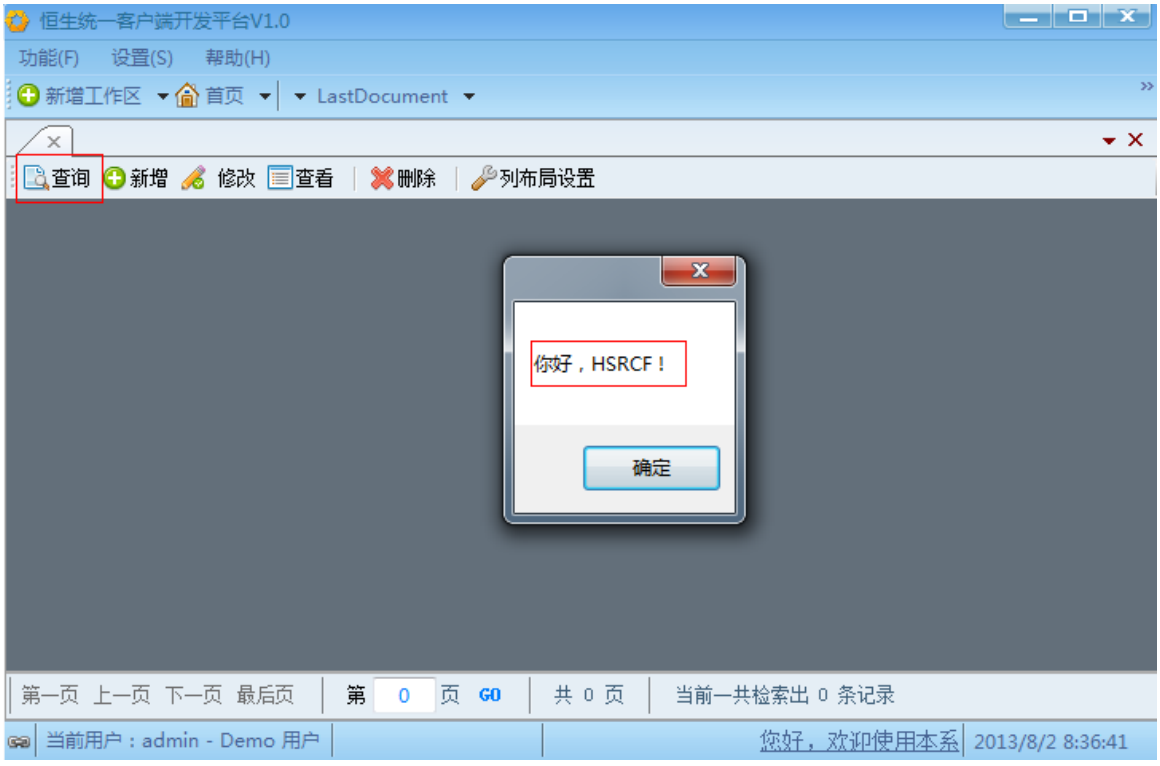
4.) 在这个设计视图中，双击“查询”工具按钮，Visual Studio 将默认切换到代码视图，并新增查询按钮单击事件的处理方法，在该方法中将弹出“**你好，HSRCF!**”消息，如下所示。

```
/// AddInPart特性各个参数说明如下：
/// Name代表插件名：必填且命名具备系统唯一性，否则框架无法找到插件；
/// Group代表插件所属分组或分类：选填；
/// Author代表插件开发人员：选填；
/// Version代表插件版本：选填；
/// Description代表插件描述信息：选填；
/// Icon代表插件使用的图标（在工作区中显示）：选填。
/// </remarks>
[AddInPart("HelloHSRCFView", "Hundsun", "Author", "1.0.0", "提供开发模版样例详细说明", "")]
public partial class HelloHSRCFView : AddInPartBase
{

    构造函数

    私有属性

    公共属性

    私有方法

    公共方法

    事件处理

    private void toolBtnQuery_Click(object sender, EventArgs e)
    {
        MessageBox.Show("你好，HSRCF！");
    }

    其他
}
```

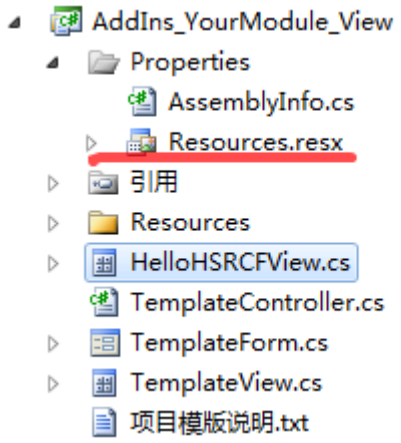
5.) 重新生成此项目，然后运行主应用程序，或者直接在 Visual Studio 中启动调试模式(默认快捷键 F5)。在登录进入开发平台主界面后，选择“设置>工作区>新建”菜单项，在弹出的对话框中，可以看到插件 HelloHSRCFView，如下图所示。



6.) 选择 HelloHSRCFView 插件，单击“确定”，可以打开该插件；然后单击查询按钮，将弹出“**你好，HSRCF！**”消息框，如下图所示。



7.) 观察 Hello HSRCF 插件，会发现它没有标题信息，可以通过项目中 Properties 文件夹下的 Resources.resx 的资源文件添加标题。

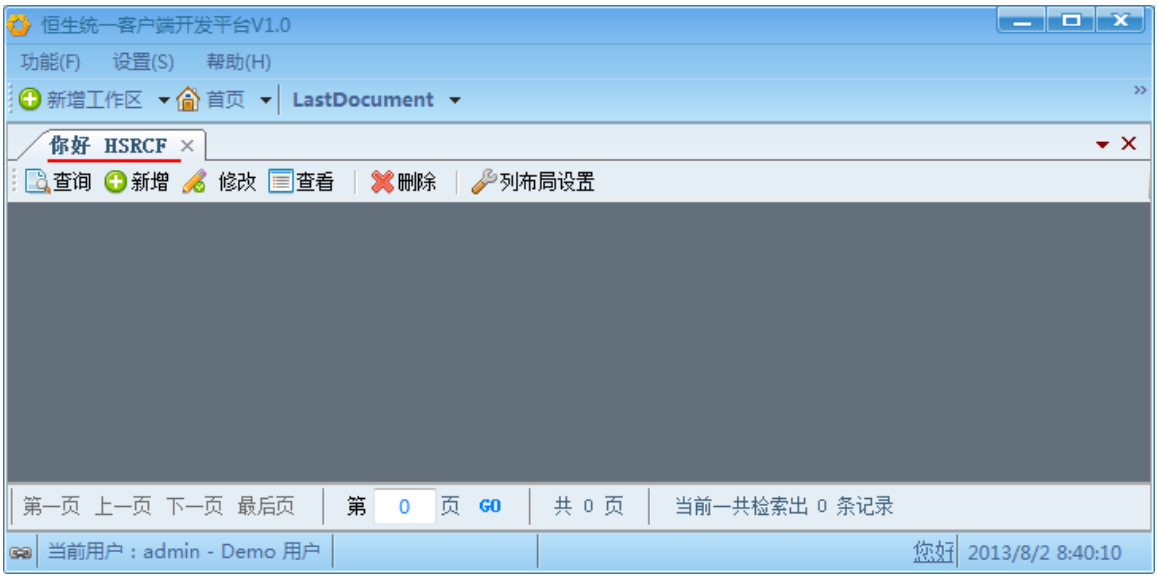


添加一个字符串资源，名称为：HelloHSRCFView，值为“你好 HSRCF”。

名称	值
HelloHSRCFView	你好 HSRCF

字符串资源的名称必须是插件的类名，例如创建的插件类名为 HelloHSRCFView，在资源文件中添加的名称必须为 HelloHSRCFView，否则无法显示插件的标签名。而资源的值可以为任意有意义的值，例如“你好，HSRCF”。

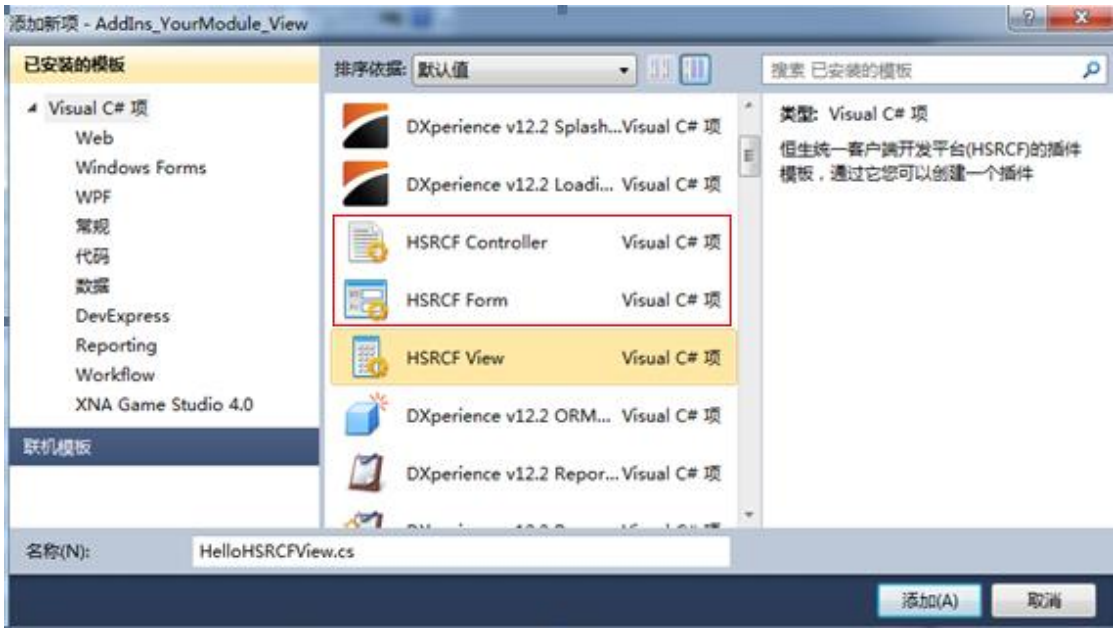
重新调试程序，将看到如下界面。



至此，一个简易的插件开发完成。

3.4 新建窗体和控制类

通过新建插件章节介绍可知，可以使用平台提供的模板来实现快速新建 HSRCF 窗体和 HSRCF 控制类。



3.5 开发约束

平台是通过提供 Visual Studio 模板来实现 HSRCF 插件的快速开发的，因此开发人员必须了解开发插件的相关约束。

3.5.1 插件项目的输出目录

插件项目生成的 dll（动态链接库）的输出路径必须为开发平台运行时执行目录下的 AddIns 子文件夹。如果通过模板创建插件项目，将自动指定为正确的目录。如果不设置为该路径，调试时会出现找不到当前所开发插件的问题。

假如开发平台运行时执行目录为
C:\Works\HSRCF\Hundsun.UI\Hundsun.Framework.Platform\bin\Debug\

那么插件项目的输出目录则为
C:\Works\HSRCF\Hundsun.UI\Hundsun.Framework.Platform\bin\Debug\AddIns\

3.5.2 基类继承和特性约束

为了便于各个业务应用的管理以及特定的处理，**建议业务应用构建自己的基类**，这些基类必须从平台相关基类中继承。

插件基类和特性约束

插件必须从 Hundsun.Framework.UIFrame.AddInPartBase 类继承而且必须标注 AddInPart 特性，示例如下：

```
[AddInPart("HelloHSRCFView", "模版", "Author", "1.0.0", "", "Template")]
public partial class HelloHSRCFView : AddInPartBase
```

窗体基类

窗口类，例如新增、修改等窗口必须从 Hundsun.Framework.UIFrame.AddInFormBase 类继承，示例如下：

```
public partial class HelloHSRCFForm : AddInFormBase
```

4 插件生命周期

HSRCF 内的插件目前都是以 Windows Form 的用户控件形式展现，合并用户控件本身的生命周期和插件公开的接口，完整的插件生命周期如下图所示。

顺序号	周期	描述
1	插件构造函数	构造插件，实例化。
2	OnLoad 方法或 Load 事件	插件第一次加入容器时触发。
3	RunAddInPart 方法	每次插件从隐藏状态变为显示状态时调用。
4	自定义 UI 事件处理过程	该过程由业务插件控制，业务操作均在此过程实现。
5	CloseAddInPart 方法	插件关闭时调用接口，目前关闭是假关，内存中还存在，以便用户再找回之前的插件。
6	销毁事件(Disposed)	插件彻底销毁。

从上表我们可以看出，可以在 OnLoad 事件中做一次性初始化任务；也可以重写 RunAddInPart 中来初始化每次显示都需要刷新的内容，而重写 CloseAddInPart 方法处理隐藏时的事件，如果需要彻底销毁插件时做处理，可以绑定 Disposed 事件。

5 工作区

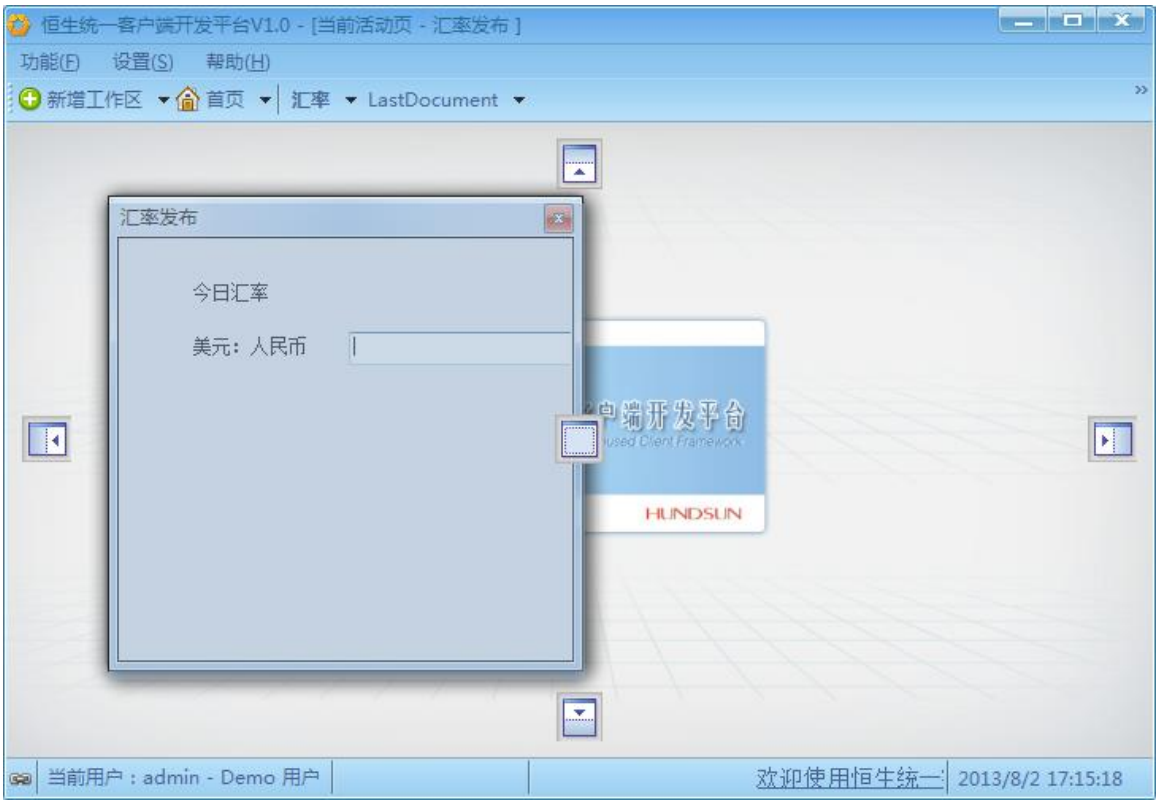
5.1 工作区管理

5.1.1 概述

工作区常指业务操作的区域，它由零个或多个插件组成。在 HSRCF 中，可以对工作区进行新建、保存、打开、关闭和建立联动（插件联动关系是指两个插件之间的发布/订阅关系）等操作；还可以针对工作区设置工作区内插件的直接联动关系，采用的工作区功能在“**设置->工作区**”菜单项下。

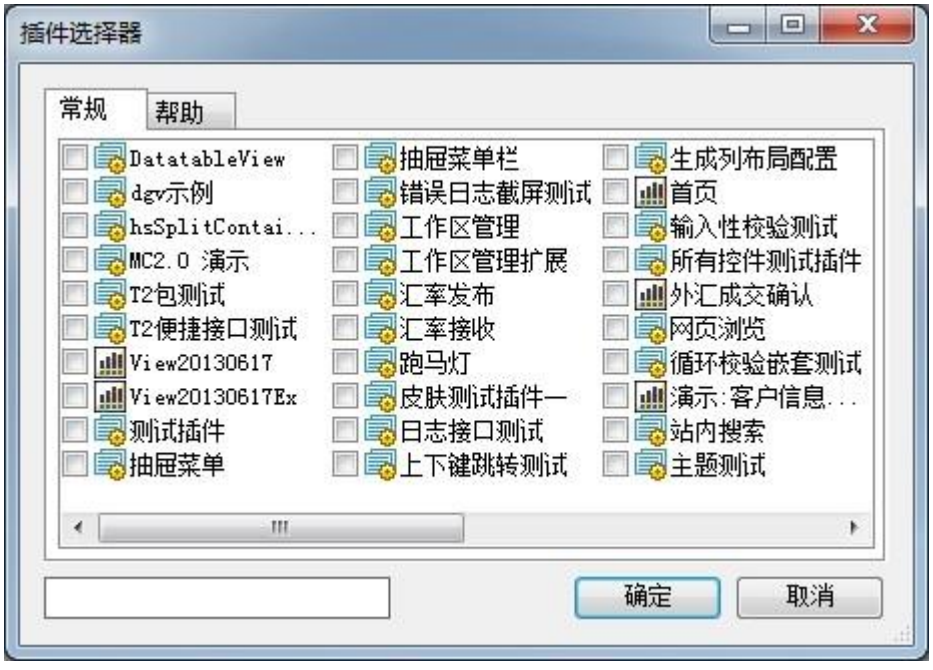
5.1.2 设计

设计功能主要是控制活动工作区是否实现可编辑状态(目前编辑状态下用户可以调整插件的布局)，如下图所示。



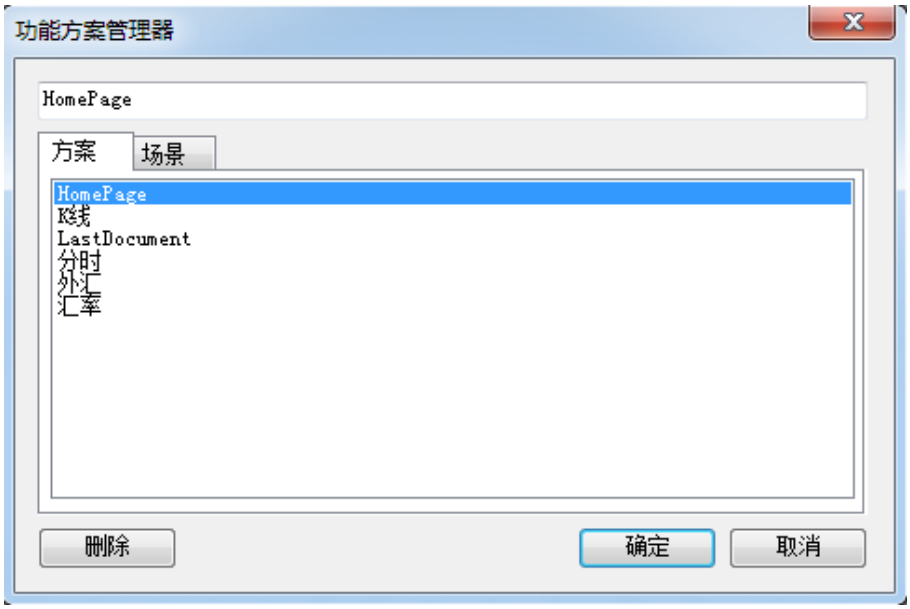
5.1.3 新建

新建功能主要是在插件选择器（如下图）中选择其中一个或多个插件，点击“确定”按钮，在主窗口的活动区域就能打开相关的业务插件面模块。



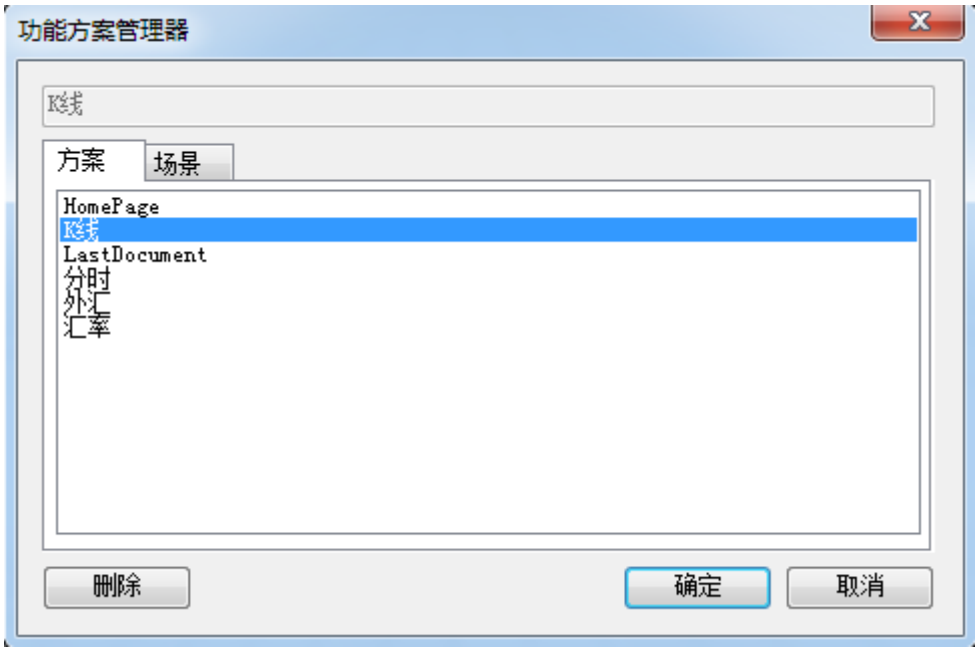
5.1.4 保存

保存工作区是把在活动工作区内显示的插件布局信息和联动信息等保存为一个配置文件，它将被保存在运行时输出目录下的 FunctionSchemes 文件夹下。保存工作区如下图所示。



5.1.5 打开

打开工作区是从已保存的工作区文件夹中显示所有的配置方案，打开工作区如下图所示。



选中其中的一个工作区配置方案后，点击“确定”按钮，可以在主窗口的活动区域打开这个工作区。

5.1.6 关闭

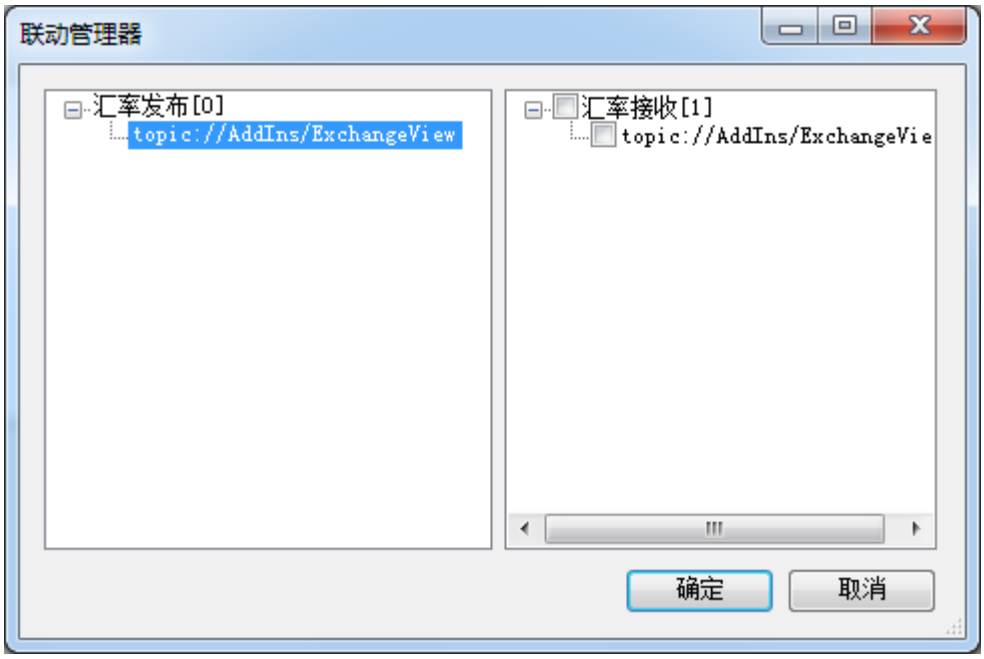
关闭工作区主要是把在活动工作区内显示的插件全部关闭，彻底释放。

5.1.7 编辑-添加插件

打开添加插件功能是在当前活动工作区显示基础上再继续添加相关的业务插件，添加插件如下图所示。

5.1.8 编辑-建立关联

建立关联功能是在活动工作区显示的插件之间建立联动关系，如下图所示。



5.2 插件联动

5.2.1 概述

联动是指两个插件之间可以建立关联关系，例如一个插件发布一个消息“你好”，另外一个插件能自动收到该消息。

5.2.2 联动开发

插件间联动关系的开发主要用到 LinkageEventPublication 和 LinkageEventSubscription 特性。这两个特性用于在两个独立开发的插件之间建立发布订阅式的联系，需要添加 Hundsun.Framework.AddIn.dll 引用。

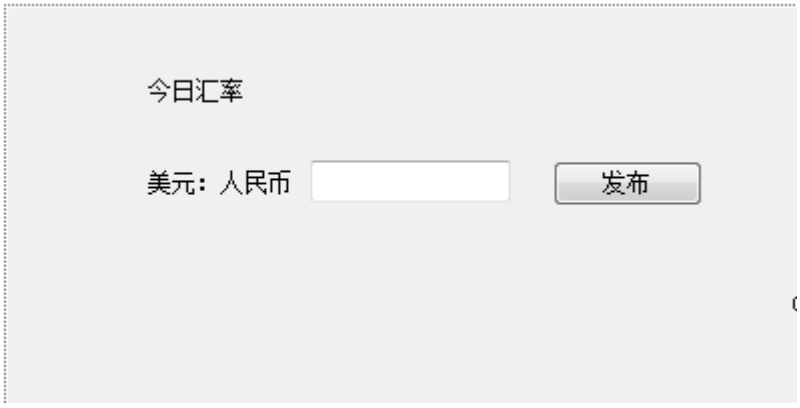
本文通过一个案例来说明此特性的用法，以下为开发步骤：

1.) 假设要建立一个有关汇率的插件项目，取名为 AddIn_Exchange_View，放在 Hundsun.AddIns 目录下。注意要修改生成路径为运行时执行目录下的 AddIns 目录下。

2.) 在此目录下建立两个用户控件，一个发布当日汇率的控件，命名为 ExchangePubView，另一个是接收并显示汇率的控件，命名为 ExchangeSubView。从 AddInPartBase 继承并且加上 AddInPart 特性，使它们成为插件模块：

```
[AddInPart("汇率发布", "Hundsun", "me", "1.0.0", "测试 Demo")]
[AddInPart("汇率接收", "Hundsun", "me", "1.0.0", "测试 Demo")]
```

3.) 在 ExchangePubView 控件上拖出相应控件，用于发布汇率。发布按钮命名为 btnPub，汇率输入文本框命名为 txtExchange。



4.) 在 ExchangePubView 控件的代码视图中，声明一个事件。在其上加入 LinkageEventPublication 特性：

```
[LinkageEventPublication("topic://exchangeInfo")]
public event EventHandler<DataEventArgs<string>> ExchangePub;
```

在其他业务插件中，所有订阅了主题名为"topic://exchangeInfo"的方法都会和 ExchangePub 事件自动关联起来。当 ExchangePub 事件被触发时，那些订阅的方法将会被触发，这样就解耦了插件间的调用联系。在 ExchangeSubView 控件中添加被触发的方法。

5.) 双击发布按钮，在生成的响应 Click 事件的函数中添加事件触发：

```
if(null != ExchangePub)
{
 ExchangePub(this， new DataEventArgs<string>(txtExchange.Text));
}
```

将汇率输入框 txtExchange 的内容作为参数发送给所有订阅了主题的方法。（这里没有验证输入的合法性，只做演示用）

6.) 在 ExchangeSubView 控件上拖出相应控件，用来接收和显示汇率。其中显示汇率文本框取名为：txtExchange

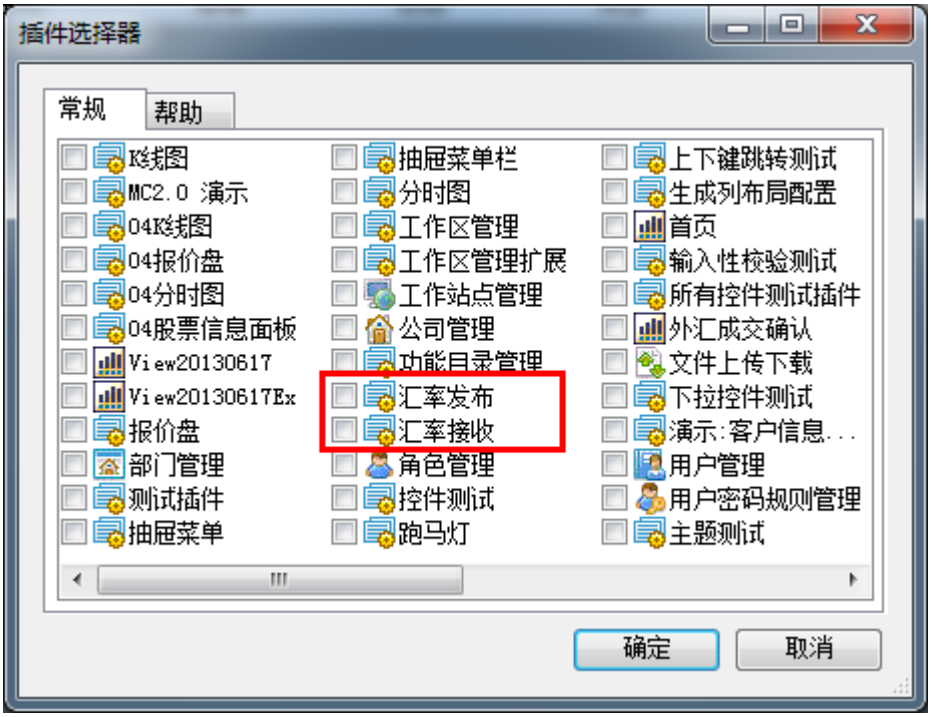
7.) 在 ExchangeSubView 控件的代码视图中，添加相应的订阅主题的方法：

```
[LinkageEventSubscription("topic://exchangeInfo")]
public void OnExchangeInfoReceived(object sender， DataEventArgs<string> e)
{
txtExchange.Text = e.Data;
```

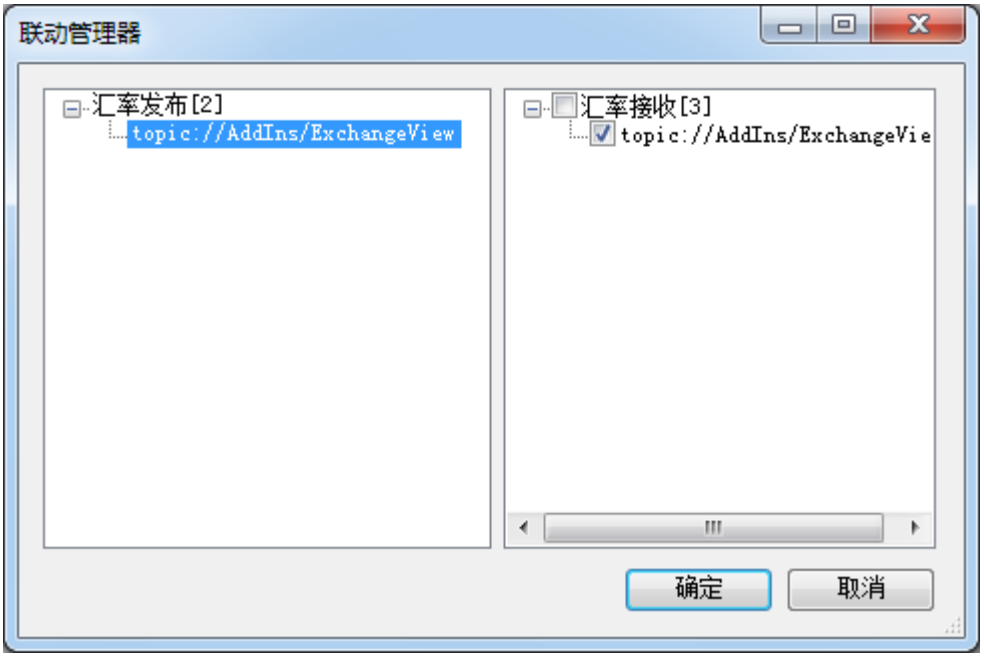
```
}
```

特性表示订阅相应主题，获取传输过来的数据，显示在汇率的文本框上。（注意，订阅的主题最好声明成一个只读的变量，而不是硬编码在程序中，这里只做演示）

8.) 生成此汇率插件的项目。启动最终应用程序，登录主界面以后，点击“设置>工作区>新建”菜单项，可以看到提供的这两个插件。

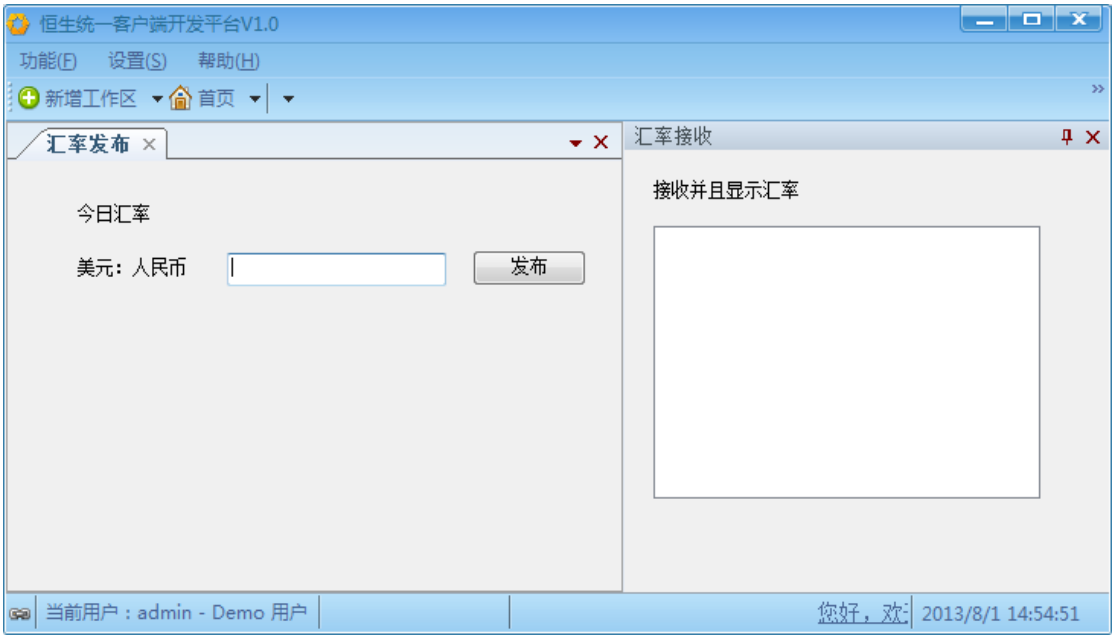


9.) 选中“汇率发布”和“汇率接收”后，单击“确定”按钮，系统会弹出联动管理器对话框，如下图所示。

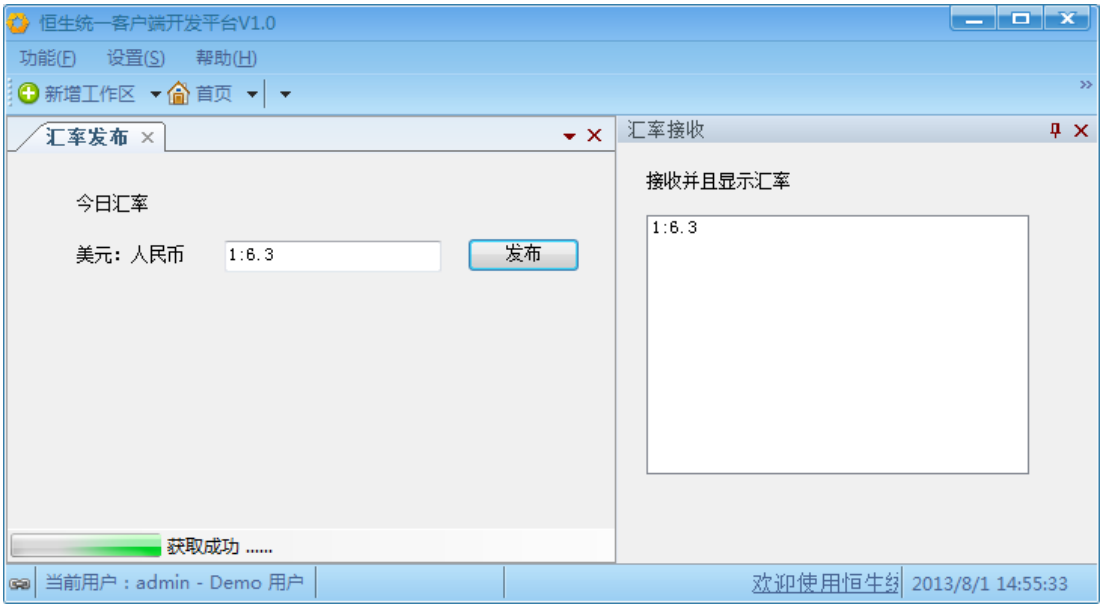


系统自动检测到了两个插件之间有联动关系。左边表示发布的主题，右边表示订阅的主题。勾选订阅的主题，单击“确定”后就完成了这两个插件的联动。

10.) 重新打开两个插件的页面，通过拖动将其调整成如下排版。



在汇率发布中填入汇率，点击“发布”，在汇率接收中就可以接收到此汇率，如下图所示。



一个简单的插件之间的发布和订阅关系就建立了。

发布和订阅的关系可以是一对多的，即一个发布者可以有多个订阅者。

6 自定义框架页

框架页即主窗体，从工作区章节可知，工作区是可以自定义的。从以上章节的相关示例图中可以看到，框架页中还包含菜单栏、工具栏和状态栏，这三栏都是可以自定义的，可以实现隐藏和配置栏目、更换 Logo 等。

6.1 菜单栏

在 HSRCF 中，主菜单栏的功能主要有以下分类：

- 打开单个插件
- 自定义菜单项

6.1.1 打开单个插件

如果需要在菜单栏中添加菜单按钮以打开某个插件，可以通过在配置文件中添加配置实现。

例如要配置一个菜单来打开Hello HSRCF插件，操作步骤如下所示：

1.) 在开发平台运行时执行目录下，打开 System 目录，其中有两个分别以语言区域命名的文件夹：en、zh-CN，这两个目录下有相应语言下的菜单栏配置选项，打开 zh-CN 文件夹下的 MainMenuStrip.xml 配置文件。

2.) 在 Navigation 标签下，配置一个菜单项：

```
<Item Label="测试">
<Item Label="插件 HelloHSRCF" CommandType="2" CommandName =
"HelloHSRCFView" ></Item>
</Item>
```

Label 表示菜单项的显示名称，CommandType 表示菜单项的类型，CommandName 是在开发插件时在 AddInPart 特性中加入的插件名。在 Hello HSRCF 插件中是：

```
[AddInPart("HelloHSRCFView", "HundSun", "me", "1.0.0", "Just a Demo")]
```

3.) 如果要在英文环境中也有相应的菜单项，那么要在 en 目录下的 MainMenuStrip.xml 文件中配置相应的菜单项，把相应的 Label 内容改成英文即可。

4.) 运行程序，可以看到相应的菜单项。

5.) 点击菜单项后可以打开相应的插件。

6.1.2 自定义菜单项(如打开工作区)

开发自定义功能的菜单项需要使用 AddInToolStripItem 特性，此特性主要用来定制框架中的菜单栏、工具栏和状态栏的每个子项，需要添加 Hundsun.Framework.AddIn.dll 引用。

假设要在菜单栏上添加一个菜单选项“测试”，其下拉菜单中有一个名为“方案 1”的选项可以选择。

1.) 用新建插件项目的方式新建 AddIn_MenuStripItem 的项目，请注意新建插件的注意事项。

2.) 新建一个用户控件，命名为 HelloMenuStripItem，使其继承自 ToolStripMenuItem，可以在其中添加单击等事件的响应方法。例如，弹出 Hello HSRCF 插件。

3.) 为此控件添加 AddInToolStripItem 特性。

```
[AddInToolStripItem("方案 1")]
   public partial class HelloMenuStripItem : ToolStripMenuItem
```

4.) 生成此插件。此时插件项目会在平台运行时执行目录下的 AddIns 文件夹下生成名为 AddIn_MenuStripItem.dll 的动态库。

5.) 在开发平台的运行时执行目录下，打开 System 目录，其中有两个分别以语言区域命名的文件夹：en、zh-CN，这两个目录下有相应语言下的菜单栏配置选项，打开 zh-CN 文件夹下的 MainMenuStrip.xml 配置文件。

6.) 在 Navigation 标签下，配置一个菜单项：

```
 <Item Label="测试">
   <Item Label="方案 1" CommandType="3" CommandName="AddIn_MenuStripItem.dll,
Hundsun.Framework.AddIns. HelloMenuStripItem "></Item>
   </Item>
```

其中 commandName 中，AddIn_MenuStripItem.dll 表示第 2 步生成的动态库，逗号后面的带命名空间的类表示动态库中的对应的菜单项类。

7.) 运行应用程序后，可看到菜单栏目中的选项。

如果需要在其他语言中提供此菜单项目，那么就在 System 目录下的其他语言文件夹的配置文件中添加此菜单项目。

6.2 工具栏

由HSRCF 界面章节可知，工具栏主要有主窗体工具栏和插件/窗体内工具栏。

6.2.1 主窗体工具栏

主窗体工具栏的作用范围是整个应用程序，所以一般主窗体工具栏上的工具项（按钮、下拉样式按钮等）主要有全局搜索框、主页快捷按钮、插件页刷新等。而对于每个应用程序的主窗体工具栏都有独特的工具项，因此各个应用程序需要自定义相应的工具项，下面将叙述如何自定义主窗体工具栏和工具项。

一般主窗体工具栏的效果如下图所示。

主窗体工具栏配置文件列表

下表列出了主窗体工具栏使用到的配置文件。

配置文件名	作用	路径
Tools.xml	用于指定**主窗体工具栏**用到的工具项配置文件名(目前文件名指定为ToolStripObject)。这里只需要配置 Item 节点的 Key 值即可，该值为文件名。	中文环境： 平台运行时执行目录下的 \System\zh-CN\ 英文环境： 平台运行时执行目录下的 \System\ en\
ToolStripObject.xml	从 Tools.xml 的作用描述中可知它包含**主窗体工具栏**的工具项的配置信息。配置信息用于动态构建工具项加入工具栏中。	

主窗体工具栏可见性

如果需要使主窗体工具栏不可见，只要将 Tools.xml 中的 **Item 节点清空或全部注释**即可。

Tool.Xml 的配置如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<ToolStripBox xmlns="http://tempuri.org/ToolStripBox.xsd">
  <Item Key="ToolStripObject"/>
</ToolStripBox>
```

隐藏该主窗体工具栏，可以如下配置：

```
<?xml version="1.0" encoding="utf-8" ?>
<ToolStripBox xmlns="http://tempuri.org/ToolStripBox.xsd">
 <!--<Item Key="ToolStripObject"/>-->
</ToolStripBox>
```

主窗体工具栏多行并排

如果要多一行工具栏就多加一个 Item 元素（Tool.xml 配置文件中），多两行就多加两个 Item 元素。

例如，Tool.Xml 原始配置信息如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<ToolStripBox xmlns="http://tempuri.org/ToolStripBox.xsd">
 <Item Key="ToolStripObject"/>
</ToolStripBox>
```

如果需要两行，完全可以如下配置：

```
<?xml version="1.0" encoding="utf-8" ?>
<ToolStripBox xmlns="http://tempuri.org/ToolStripBox.xsd">
 <Item Key="ToolStripObject"/>
<Item Key="ToolStripObject"/>
</ToolStripBox>
```

这样就生成两行完全相同的工具栏。并排生成的顺序即配置文件中 Item 元素的顺序，如下图所示。

自定义主窗体工具项

对于主窗体工具栏，自定义其工具项只需要两步：

1.) 创建工具项类

- **方式一**：从 ToolStripItem 的派生类中继承，这种方式需要给类头标注 AddInToolStripItem 特性。

  可以继承 hsToolStripMenuItem 创建菜单项样式工具项

  可以继承 hsToolStripButton 创建按钮样式的工具项

  可以继承 hsToolStripDropDownButton 创建按钮方式弹出下拉的工具项

  例如：

```
/// <summary>
/// 关闭插件页
/// </summary>
[AddInToolStripItem("CloseDocument")]
public partial class CloseDocumentToolStripItem : hsToolStripButton
```

- **方式二：**创建一个插件用于承载工具项

  创建一个插件类，记得标注 **AddInPart** 特性。

```
[AddInPart("工具栏宿主", "Tools", "Hundsun", "1.0", "自定义工具项")]
public partial class ToolView : UserControl
```

2.) 配置工具项信息（配置文件为 **Tools.xml** 文件中 Item 元素指定的文件中）

下面是工具项的配置信息。

```
<?xml version="1.0" encoding="utf-8" ?>
<ToolStripObject BackColor="LightGray" BackgroundImage="toolbar"
xmlns="http://tempuri.org/ToolStripObject.xsd">
  <Item PersistString="Hundsun.Framework.Tools.dll,
Hundsun.Framework.AddIns.ToolView" ControlHost="true"/>
  <Item PersistString="Hundsun.Framework.Tools.dll,
Hundsun.Framework.AddIns.CloseDocumentToolStripItem" RightAlignment="true"/>
</ToolStripObject>
```

配置元素或属性	说明
Item	一个工具项/一个元素项配置文件名。
PersistString	该属性用于动态构建工具项。格式必须为“**工具项所在动态库，类全名”**。
ControlHost	该属性用于表示是否插件承载。如果是方式二创建工具项，可以设置 ControlHost="**true**"。
RightAlignment	表示是否右对齐。

针对创建工具项的两种方式，配置方式也分如下两种：

对于方式一，一般配置方式如下：

```
<Item PersistString="Hundsun.Framework.Tools.dll,
Hundsun.Framework.AddIns.CloseDocumentToolStripItem" RightAlignment="true"/>
```

对于方式二，一般配置方式如下：

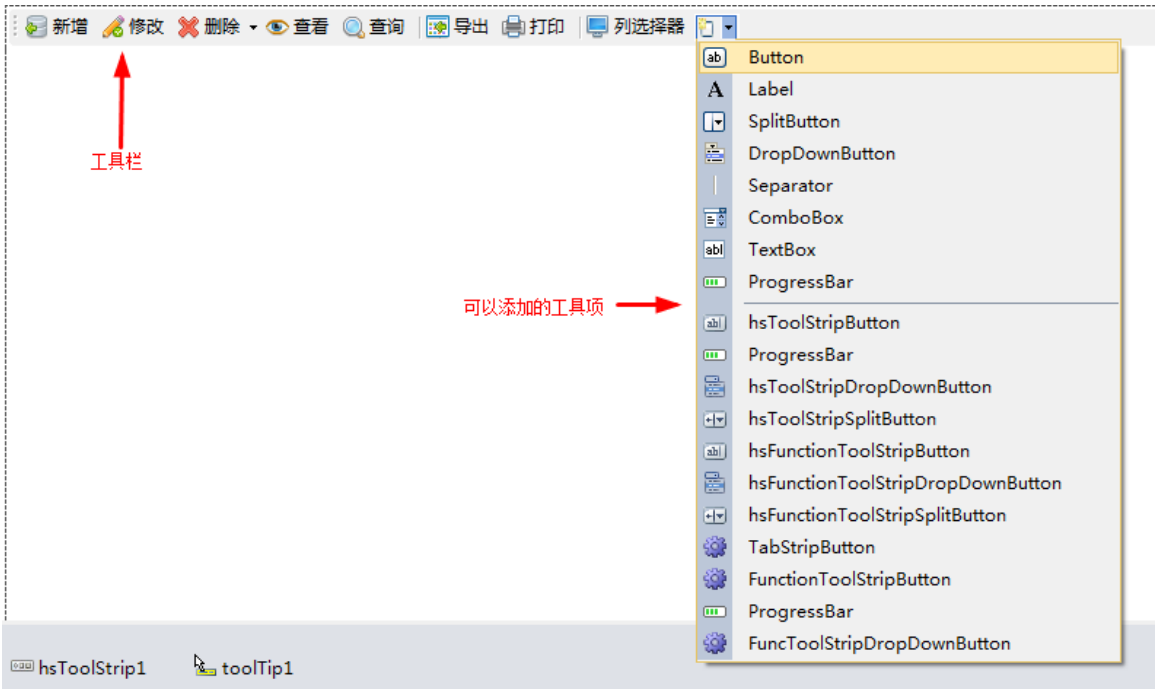
```
<Item PersistString="Hundsun.Framework.Tools.dll,
```

```
Hundsun.Framework.AddIns.ToolView" ControlHost="true"/>
```

6.2.2 插件/窗体内工具栏

插件/一般窗体内的工具栏与常用的工具栏的使用方式是一致的，都是将工具栏控件如hsToolStrip（它是从 ToolStrip 继承）拖入插件或窗体内，然后添加必要的工具项即可。

原则上，尽量采用框架提供的工具项，框架提供的工具项均以 hs 打头，这样使用便于框架控制主体样式。



6.3 状态栏

对于状态栏，主要是指主窗体的状态栏，如果需要在插件内和窗体内使用也可以。

6.3.1 主窗体状态栏

状态栏的可见性、多行并排以及自定义工具项的方式都是一样的，只是配置文件名不同，其配置方式和节点名也是一样的。

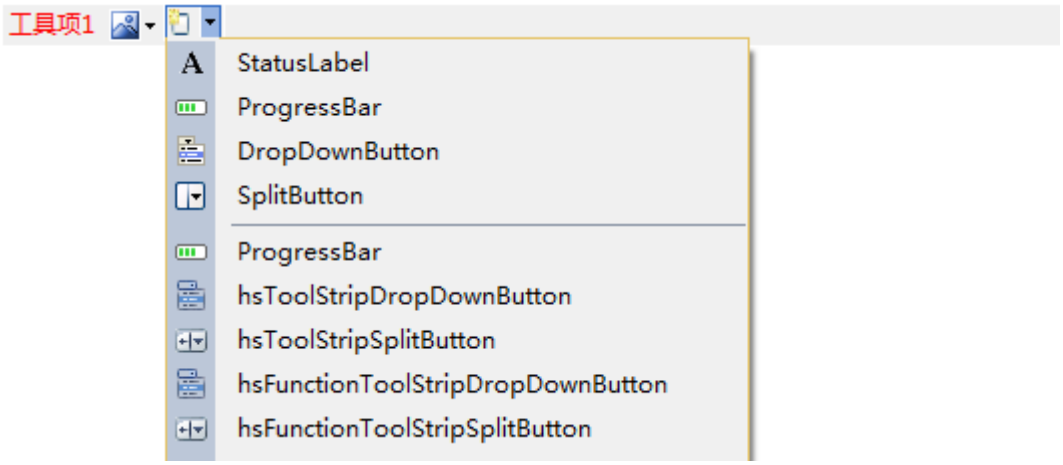
下面是状态栏使用到的配置文件。

配置文件名	作用	路径
Status.xml	用于指定**主窗体状态栏**的工具项配置文件名(目前文件名指定为 StatusObject)。 这里只需要配置 Item 节点的 Key 值即可，该值为文件名。	中文环境： 平台运行时执行目录下的\System\zh-CN\ 英文环境： 平台运行时执行目录下的\System\ en\
Status Object.xml	从 Status.*xml* 的作用描述中可知它包含**主窗体状态栏**的工具项的配置信息。	

目前工具项的配置中 RightAlignment="true"是不起作用的。

6.3.2 插件/窗体内状态栏

插件/一般窗体内的状态栏与主窗体的状态栏的使用方式是一致的，都是将状态栏控件如 hsStatusStrip（它是从 StatusStrip 继承）拖入插件或窗体内，然后添加必要的工具项即可。原则上，尽量采用框架提供的工具项，框架提供的工具项均以 hs 打头，这样使用是为了便于框架控制主体样式。



7 客户端通信

7.1 概述

目前框架已将 T2、MC (消息中心)等通信协议封装入 Hundsun.Framework.Communication.dll 动态库中，只需添加引用该动态库即可使用。

下表是各协议的接口和实现对象。

协议	主要接口	接口实现类
T2	IT2DataHandler	T2DataHandler
MC2（消息中心）	IMC2Subscriber	MC2Subscriber
	IMC2Publisher	MC2Publisher

框架提供了一个顶层的数据管理接口 **IDBFactory**，而 T2、MC 等协议都是从这个接口中获取实例。目前该接口有一个默认实现类 **DBFactory**，为了保持该实例的唯一性，目前不建议用户自行构建该类，框架已经提供了 ICommunication 接口的实现类内部构建了该实例。

如果想详细了解通信各个协议的接口方法、属性，可以查看 chm 帮助文件。

下面说明如何获取相应的协议接口实例。

7.2 获取各协议接口实例

目前对于获取各协议接口实例，基本上是采用同样的方式进行处理的，除了获取订阅者实例。下面以 IT2DataHandler 接口为例，介绍下各协议接口实例的获取方式。

IT2DataHandler 接口实例的获取主要有以下几种方式：

- 插件中，可以通过 this.T2 属性获取接口实例。

- 窗体中，也可以通过 this.T2 属性获取接口实例。
- 如果使用无参构造函数构造窗体，框架将默认 T2 的实例，可以直接通过 this.T2 获取；如果不是使用无参构造函数构造窗体，则必须传入 T2 的接口实例，这个接口可以通过 this.T2 属性获取。
- 通过 ICommunication.DBFactory.GetT2Data() 获取。

对于其他协议，如 MC 等也是采用相同的方式处理的。

下表列出了获取接口的属性名称和获取方法。

协议和接口	插件中	窗体中	IDBFactory 的工厂方法
IT2DataHandler	this.T2 属性	构造函数传入接口实例，在窗体中使用	IDBFactory.GetT2Data()
IMC2Publisher (发布者)	this.MC2Publisher 属性(默认发布者，全局唯一) 或 this.GetMCPublisher(string **publisherName)**		IDBFactory. GetMC2Publisher()
IMC2Subscriber (订阅者)	this.GetMCSubscriber(string **subscriberName**)	-	DBFactory.GetMC2Subscriber(string **subscriberName**)

其中，参数 **subscriberName/publisherName** 为订阅者/发布者名字，**必须**全局唯一（建议和具体业务相关），获取已存在的订阅者/发布者会报错。

7.3 使用 T2 协议开发

T2 具体的接口说明可以参考《**IT2DataHandler T2 通讯接口设计和使用说明.xlsx**》文档。在这里将简述两个便捷接口的示例以及原生 T2 接口示例。

7.3.1 使用 T2 便捷接口开发

HSRCF 在 T2DataHandler 中封装了一对便捷的接口方法 GetDataTable 和 OperationData。

查询操作

可以在插件中直接使用 this.T2.GetDataTable(…)来得到查询数据，绑定到列表，再结合列布局，一个基本的查询列表瞬间就能做好。

同步接口方法描述如下：

```
/// <summary>
```

```
/// T2 获取后台传输的 DataTable 数据
/// 异常：T2CommunicationException
/// </summary>
/// <param name="fieldNames">列名字数组，入参字段列表</param>
/// <param name="values">字段值数组，入参值列表</param>
/// <param name="functionID">功能号</param>
/// <param name="timeout">超时时间，以毫秒为单位</param>
/// <param name="captions">返回列字段别名数组</param>
/// <returns>DataTable</returns>
public DataTable GetDataTable(List<string> fieldNames, List<object> values, int functionID, int timeout, List<string> captions)
```

异步接口方法描述如下：

```
/// <summary>
/// 异步获取 DataTable 数据
/// 异常：
/// T2CommunicationException
/// </summary>
/// <param name="functionNo">功能号</param>
/// <param name="callback">
/// 收到 Datatable 时的回调
/// 注意：请务必传入回调委托，否则无法收到异步应答
/// </param>
/// <param name="fieldNames">列名字数组</param>
/// <param name="values">字段值数组</param>
void AsyGetDataTable(int functionNo, OnReceivedDataTableDelegate callback, List<string> fieldNames, List<object> values);
```

对于异步，需要传入回调的方法，在该回调方法中，可以对异步应答数据做相应处理，如将数据绑定到 DataGridview 中（注意：异步处理逻辑不能太复杂、太耗时，否则将导致底层通信阻塞，必要时需要将数据拷出另起线程处理），如下所示：

```
/// <summary>
/// 收到异步应答时发生
/// 注意：
/// 异步应答出错信息将写入 T2 日志，必要时需自行查看日志
/// </summary>
/// <param name="functionNo">功能号</param>
/// <param name="dt">DataTable</param>
private void OnReceivedDataTable(int functionNo, DataTable dt)
{
    this.ShowData(dt);
}
```

```
/// <summary>
/// 显示 DataTable 数据
/// </summary>
/// <param name="dt">DataTable</param>
private void ShowData(DataTable dt)
{
    /*
        * 说明：
        * 为防止异步应答 访问插件私有资源时 发生异常，
        * 建议调用 Invoke 以委托方式执行，如下所示！
    */
    if (this.dgvData.InvokeRequired)
        this.dgvData.Invoke(new Action<DataTable>(this.ShowData), dt);
    else
    {
        this.dgvData.DataSource = dt;
    }
}
```

非查询操作（如增，删，改等）

非查询操作目前仅支持同步操作，接口方法描述如下：

```
/// <summary>
/// 用于增加、删除、修改等无返回值操作
/// 结果：true:操作成功  false:操作失败
/// 异常：T2CommunicationException
/// </summary>
/// <param name="fieldNames">列名字段数组，入参字段列表</param>
/// <param name="values">字段值数组，入参值列表</param>
/// <param name="functionID">功能号</param>
/// <param name="timeout">请求处理超时时间，以毫秒为单位</param>
/// <returns> true:操作成功  false:操作失败</returns>
public bool OperationData(List<string> fieldNames, List<object> values, int
functionID, int timeout)
```

7.3.2 使用原生 T2 接口开发

同步数据收发示例

```
        /// <summary>
        /// 同步获取数据
        /// </summary>
```

```
        /// <param name="dgvData"></param>
        public DataTable GetData()
        {
            // 在 T2Packer 池中取 T2 打包器接口（默认版本 2）
            IT2Packer packer = this.T2.GetT2Packer(T2PackVersion.PACK_VERSION2);
            // 打包
            try
            {
                packer.BeginPack();
                packer.AddField("selectText", "select * from ExchargeData");
                packer.AddField("action", T2FieldType.TYPE_INT, 10, 0, 1);
                packer.EndPack();
            }
            catch (T2PackerException)
            {
                // TODO:
                // 释放 T2 打包器接口
                t2.ReleaseT2Packer(packer);
                return null;
            }

            // 在 T2ESBMessage 池中取 T2 报文操纵接口
            IT2ESBMessage esbmsg = t2.GetT2Esbmsg();
            // 初始化报文
            esbmsg.Prepare(T2TagDef.REQUEST_PACKET, 70);
            // 设置报文包体部分数据
            esbmsg.MsgBody = packer.PackerBuffer;
            // 释放 T2 打包器接口
            t2.ReleaseT2Packer(packer);
            //同步发送
            try
            {
                t2.SynSendEsbMessage(ref esbmsg, 2000);
            }
            catch (T2CommunicationException)
            {
                // TODO:
                return null;
            }
            // 返回码或者错误码不为 0，表示业务处理异常
            if (esbmsg.ReturnCode != 0 ||
                esbmsg.ErrorNo != 0)
```

```
        {
            // TODO:
            return null;
        }
        // 业务包体为空，或者长度小于 20（此为非法包体）
        if (esbmsg.MsgBody == null || esbmsg.MsgBody.Length < 20)
        {
            // TODO:
            return null;
        }

        // 在 T2UnPacker 池中取 T2 解包器接口，解包
        IT2UnPacker unpacker = t2.GetT2UnPacker(esbmsg.MsgBody);
        t2.ReleaseT2Esbmsg(esbmsg);

        // 数据不为空 且 存在 error_no 字段（表示出错）返回
        string name = String.Empty;
        if (!unpacker.Empty &&
            ((name = unpacker.GetColName(0).ToLower()).Contains("error_no") ||
            name.Equals("errorno")))
        {
            // TODO:
            // 如：获取错误号和错误信息
            int errorNo = unpacker.GetInt("error_no");
            string errorInfo = unpacker.GetStr("error_info");

            // 释放 T2 解包器接口
            t2.ReleaseT2UnPacker(unpacker);
            return null;
        }

        DataTable dt = unpacker.ToDataTable(null);
        // 释放 T2 解包器接口
        t2.ReleaseT2UnPacker(unpacker);
        return dt;
    }
```

异步数据收发示例

```
    /// <summary>
    /// 异步获取数据
    /// </summary>
    public void AsyGetData()
```

```
        {
            // 在 T2ESBMessage 池中取 T2 报文操纵接口，同时初始化
            IT2ESBMessage esbmsg = t2.GetT2Esbmsg(T2TagDef.REQUEST_PACKET, 70);
            // 在 T2Packer 池中取 T2 打包器接口
            IT2Packer packer = t2.GetT2Packer();

            // 打包
            packer.BeginPack();
            packer.AddField("sDSName", "usersvr");
            packer.AddField("selectText", "select * from ExchargeData");
            packer.AddField("action", T2FieldType.TYPE_INT, 10, 0, 1);

            packer.EndPack();

            // 设置报文包体部分数据
            esbmsg.MsgBody = packer.GetPackBuf();

            // 释放 T2 打包器接口
            t2.ReleaseT2Packer(packer);

            // 重新挂载异步数据接收委托
            //t2.OnReceived -= callback_ReceivedCallBackHandler;
            //t2.OnReceived += callback_ReceivedCallBackHandler;

            // 异步发送
            t2.AsySendEsbMessage(esbmsg, this.callback_ReceivedCallBackHandler);
            // 释放 T2 报文操纵接口
            t2.ReleaseT2Esbmsg(esbmsg);
        }
```

异步回调方法回调后将数据绑定到列表上，如下所示：

```
      public void callback_ReceivedCallBackHandler(object sender, IT2ESBMessage
esbmsg)
        {
            if (esbmsg == null)
            {
                // TODO:
                return;
            }

            // 返回码或者错误码不为 0，表示业务处理异常
            if (esbmsg.ReturnCode != 0 ||
                esbmsg.ErrorNo != 0)
```

```
        {
            // TODO:
            return;
        }
        // 业务包体为空，或者长度小于 20（此为非法包体）
        if (esbmsg.MsgBody == null || esbmsg.MsgBody.Length < 20)
        {
            // TODO:
            return;
        }

        // 新建 T2 解包器解析 T2 业务报文（也可以通过 IT2DataHandler 获取）
        IT2UnPacker unpacker = new T2UnPacker(esbmsg.MsgBody);

        // 数据不为空 且 存在 error_no 字段（表示出错）返回
        string name = String.Empty;
        if (!unpacker.Empty &&
            ((name = unpacker.GetColName(0).ToLower()).Contains("error_no") ||
            name.Equals("errorno")))
        {
            // TODO:
            // 如：获取错误号和错误信息
            int errorNo = unpacker.GetInt("error_no");
            string errorInfo = unpacker.GetStr("error_info");

            // 释放解包器
            unpacker.Dispose();
            unpacker = null;
            return;
        }

        DataTable dt = unpacker.ToDataTable(null);
        // 释放解包器
        unpacker.Dispose();
        unpacker = null;

        this.ShowData(dt);
    }

    //显示数据
    private void ShowData(DataTable dt)
    {
```

```
            //因为回调方法不在 UI 线程，所以需要 Invoke 方式进行调用
            if (dgvData.InvokeRequired)
                dgvData.Invoke(new Action<DataTable>(ShowData),  dt);
            else
            {
                dgvData.DataSource = dt;
            }
        }
```

7.3.3 T2 通信异常约束

针对 T2 通信异常，客户端与服务器端存在如下两种方式的约束：

- 消息头方式
- Packer 方式

消息头方式

即约定服务器端将业务处理的错误相关消息放到 esbmsg 的 ErrorNo 和 ErrorInfo 两个字段。

- ErrorNo：后台业务自定义错误号
- ErrorInfo：后台业务自定义错误消息内容

注意

在设定后台业务处理错误号 ErrorNo 时，**-1，0** 和 **1** 为 T2 保留错误号，**不可占用**。

Packer 方式

即约定如果服务端业务处理发生错误，比如登录时证书错误，那么返回的二维表数据中必须遵循如下规则：

- 第一列的字段名一定要叫"errorno"或者"error_no"，第一列的字段值为证书错误的错误号(必须大于 1)。
- 第二列的字段名一定要叫"errorinfo"或者"error_info"，第二列的字段值为证书错误的提示内容。

7.4 消息订阅

7.4.1 概述

消息中心 2.0 负责接收订阅者主题订阅，并发布主题消息到主题订阅者。目前，主题消息的报文格式支持业务打包格式和非业务打包格式，消息中心服务器原样发布主题消息，订阅者需要自行解析收到的主题消息。

客户端框架提供 **IMC2Subscriber** 消息中心 2.0 订阅者接口以进行消息中心主题订阅和取消订阅，使用订阅者提供的 Subscribe，CancelSubscribe 等方法即可进行主题的订阅和取消订阅；客户端框架同时提供 **IMC2Publisher** 消息中心 2.0 发布者接口以进行消息中心主题发布，使用发布者提供的 Publish 方法即可进行消息的发布。

对于获取消息中心订阅者和发布者接口实例，请参见 获取各协议接口实例 章节。

业务开发过程中，使用最多的是 this.GetMC2Subscriber(string subscriberName)方法，该方法可以获取 IMC2Subscriber 消息订阅者接口实例。

另外，在插件内还可以通过 **this.MC2Publisher** 属性获取全局发布者或通过 this.GetMCPublisher(string publisherName)来获取具有指定名字的发布者。

建议一个插件对应一个订阅者；多个订阅者/发布者的**名字**必须不一样，名字最大长度 32 个字节。

同时，如果需要熟悉消息中心 2.0 具体的 API，请参考《消息中心 2.0 开发 API 使用指南.doc》。

注意事项

- 同一主题可以使用不同的过滤器（过滤条件的组合）进行订阅。
- 当订阅者不再需要任何订阅时，可取消订阅者所有订阅，如：

```
// 取消消息订阅者所有订阅
// this.MC2Subscriber 为 IMC2Subscriber 接口实例
if (this.MC2Subscriber != null)
{
    try
    {
        // 取消该订阅者所有订阅
        this.MC2Subscriber.CancelSubscribe();
    }
    catch { }
}
```

- 当插件不再使用订阅者时，必须显式释放订阅者，此时需要重写插件关闭事件处理方法，在该方法内释放消息订阅者，如下所示：

```
/// <summary>
/// 插件关闭时发生
/// </summary>
protected override void CloseAddInPart()
{
      // 释放消息订阅者
      if (this.MC2Subscriber!= null) //this.MC2Subscriber 为 IMC2Subscriber 接口实例
      {
          try
          {
              // 将取消该订阅者所有订阅
              this.MC2Subscriber.Dispose();
              this.MC2Subscriber = null;
          }
          catch
      {
              // TODO:
      }
      }
}
```

7.4.2 订阅消息

订阅消息方法一览

下面列出了完整参数的方法，其他重载方法请直接在编辑器中查看接口声明。

```
/// <summary>
/// 订阅主题
/// 异常
/// MC2Exception，订阅失败
/// </summary>
/// <param name="subscribeParam">订阅参数</param>
/// <param name="timeout">超时时间</param>
/// <param name="bizUnPack">业务校验时，失败返回的业务错误信息.
/// 如果订阅成功没有返回，输出参数，需要外面调用 Dispose 释放
/// 如果接受业务校验的错误信息，写法如下：
/// IT2UnPacker bizUnPack = null;
/// Subscribe(..., out bizUnPack);
/// 最后根据返回值，如果是失败的就判断 bizUnPack 是不是 null.
```

```
/// 最后错误信息获取完之后， 释放
/// bizUnPack.Dispose();</param>
/// <param name="bizPack">业务校验需要增加的业务字段以及值，
/// 没有就根据过滤属性作为业务校验字段</param>
///<returns>返回值大于 0，表示当前订阅成功的订阅标识，外面要记住这个标识和订阅项之间的映射关系，
/// 这个标识需要用于取消订阅和接收消息的回调里面.
/// 返回其他值表示订阅失败，可根据错误号获取错误信息</returns>
int Subscribe(IMC2SubscribeParameter subscribeParam, uint timeout, out IT2UnPacker bizUnPack, IT2Packer bizPack);
```

订阅消息示例

```
// 消息中心订阅者
// 获取消息订阅者的方法来自插件的基类 AddInPartBase，您必须传入全局唯一的订阅// 者名字!

// 插件的构造函数
public AddInConstructor()
{
    MC2Subscriber = this.GetMC2Subscriber("订阅者名字");
}
//订阅者
private IMC2Subscriber MC2Subscriber = null;

// 订阅消息
private void SubTopic()
{
      // 设置订阅属性，如添加过滤条件等
      IMC2SubscribeParameter subparam = new MC2SubscribeParameter();
      // 主题名
      subparam.TopicName = "secu.hq";
      // 补缺订阅，默认 false
      subparam.FromNow = false;
      // 覆盖订阅，默认 false
      subparam.Replace = false;
      // 发送间隔，默认 0
      subparam.SendInterval = 0;
      // 附加数据，默认 null
      string appendData = "append";
      subparam.AppendData = Encoding.Default.GetBytes(appendData);

      // 添加过滤条件
```

```
        /*
         * 过滤条件请参考主题定义
         */
        subparam.SetFilter("exchange_type", "A");
        subparam.SetFilter("stock_code", "600570");

        // 使用指定订阅条件订阅主题
        try
        {
            // 订阅，无异常发生表示订阅成功
            this.MC2Subscriber.Subscribe(subparam, 3000);
            MessageUtility.Show(string.Format("主题: {0} 订阅成功",
subparam.TopicName));
        }
        catch (MC2Exception ex)
        {
            MessageUtility.Show(string.Format("主题: {0} 订阅失败\r\n 错误码={1}, 错
误信息={2}", subparam.TopicName, ex.ErrorNo, ex.ErrorInfo));
        }
        finally
        {
            subparam.Dispose();
        }
}
```

7.4.3 取消订阅

取消订阅方法一览

```
/// <summary>
/// 取消指定主题的订阅
/// 异常
/// MC2Exception，取消失败
/// </summary>
/// <param name="subscribeID">消息对应的订阅标识，这个标识来自于 Subscribe 方法的返回
</param>
void CancelSubscribe(int subscribeID);
/// <summary>
/// 取消指定主题的所有订阅
/// 异常
/// MC2Exception，取消失败
/// </summary>
```

```
/// <param name="topicName">主题名</param>
void CancelSubscribe(string topicName);
/// <summary>
/// 取消所有主题的所有订阅
/// 异常
/// MC2Exception，取消失败
/// </summary>
void CancelSubscribe();
```

取消订阅示例

```
// 取消订阅
private void CancelTopic()
{
      // 将要取消订阅的主题名
      string topicName = "secu.hq";

      /** 取消特定主题的所有订阅
       * 注意：
       *    调用该方法将取消主题的所有订阅
       *    也可以调用 CancelSubscribe(int subscribeID)取消特定订阅 ID 的订阅
       *    详见接口定义
       * */
      try
      {
          // 取消订阅，无异常发生表示成功取消订阅
          this.MC2Subscriber.CancelSubscribe(topicName);
          MessageBox.Show(String.Format("主题{0} 取消订阅成功", topicName));
      }
      catch (MC2Exception ex)
      {
          // TODO:
          MessageBox.Show(String.Format("主题{0} 取消订阅失败！ ({1})", topicName,
ex.ToShortString()));
      }
}
```

7.4.4 处理收到的消息

如果希望消息订阅后能处理服务器推送回来的消息，就必须挂接消息发布事件或委托方法。

挂接事件：

```
//可以将下面这段代码写在插件的 Load 事件中，以便插件初始化后就能收到消息发布消息
this.MC2Subscriber.OnPublished += new
OnPublishedEventHandler(OnPublishedHandler);
```

处理消息时候需要在以下的方法中处理收到的消息：

```
// 处理收到的订阅消息
/*
 * 返回的二进制数据是一个发布的业务数据，如果是 T2 打包器的数据就需要
 * 使用 IT2UnPacker 来解析。SubscribeRecvData 是除了业务数据之外，
 * 返回订阅相关的数据，主要有：
 * 过滤数据（使用 IT2UnPacker 解析），附加数据，主题名字；
 * subscribeID 是这个消息对应的订阅标识
 * ------------------------------------------------------------
 * 注意：
 * 回调是会占用底层的线程处理，消息处理方法需要尽快返回，
 * 也即该方法不可以过于复杂，否则会影响底层的接收性能
 */
void OnPublishedHandler(int subscribeID, byte[] data, SubscribeRecvData
appendData)
{
  // TODO: 尽快处理接收到的消息内容后返回
  MessageBox.Show(string.Format("收到订阅消息\r\n 主题名：{0}， 订阅 ID：{1}\r\n 消息长
度：{2}\r\n",
        appendData.TopicName, subscribeID, data != null ? data.Length : 0));

  // 返回的二进制数据是一个发布的业务数据，如果是打包器的数据
  // 就需要使用 IT2UnPacker 来解析，如：
  if (data != null && data.Length >= 20)
  {
      IT2UnPacker unpacker = new T2UnPacker(data);
      // 解数据
      while (!unpacker.EOF)
      {
          // TODO: 提取数据
          // ...
          unpacker.Next();
      }
  }
  // TODO: 其他消息处理逻辑
  // ...
  return;
}
```

说明

返回的二进制数据是一个发布的业务数据，如果是 T2 打包器的数据就需要使用 IT2UnPacker 来解析。SubscribeRecvData 是除了业务数据之外，返回订阅相关的数据，主要有：过滤数据（使用 IT2UnPacker 解析），附加数据，主题名字；subscribeID 是这个消息对应的订阅标识。

回调是会占用底层的线程处理，消息处理方法需要尽快返回，也即该方法不可以过于复杂，否则会影响底层的接收性能。

如果想取消已经挂接的消息发布事件或委托，可以使用如下方式：

this.MC2Subscriber.OnPublished -= new OnPublishedEventHandler(OnPublishedHandler);

7.4.5 消息发布

发布消息方法一览

下面列出了完整参数的方法，其他重载方法请直接在编辑器中查看接口声明。

```
/// <summary>
/// 非业务打包格式的内容发布接口，一般二进制格式报文发布
/// 异常
/// MC2Exception，发布失败
/// </summary>
/// <param name="topicName">主题名字，不知道名字就传 null</param>
/// <param name="filter">过滤条件，需要上层自己指定，否则默认没有过滤条件</param>
/// <param name="data">具体的内容</param>
/// <param name="timeout">超时时间</param>
/// <param name="bizUnPack">业务校验时，失败返回的业务错误信息，
/// 如果发布成功没有返回，输出参数，需要外面调用 Dispose 释放
/// 如果接受业务校验的错误信息，写法如下：
/// IT2UnPacker bizUnPack = null;
/// Publish(..., out bizUnPack);
/// 最后根据返回值，如果是失败的就判断 bizUnPack 是不是 null.
/// 最后错误信息获取完之后， 释放
/// bizUnPack.Dispose();</param>
/// <param name="addTimeStamp">是否添加时间戳，配合单笔性能查找；默认不添加</param>
void Publish(string topicName， IMC2Filter filter， byte[] data， int timeout，
    out IT2UnPacker bizUnPack， bool addTimeStamp);
```

消息发布示例

```
// 消息中心发布者
// 该属性来自插件的基类 AddInPartBase， 可以直接用 this.MC2Publisher 方式使用
public IMC2Publisher MC2Publisher {get;}
```

```
// 消息发布
private void Publish()
{
  // 主题名
  string topicName = "secu.hq";
  // 构造发布过滤条件
  IMC2Filter filter = new MC2Filter();
  filter.SetFilter("exchange_type", "ABC");
  filter.SetFilter("stock_code", "600570");

  // 构造发布消息
  IT2Packer packer = this.t2.GetT2Packer();
  packer.BeginPack();
  packer.AddField("stockcode", "600570");
  packer.AddField("stockname", "hundsun");
  packer.AddField("openprice", "15.8");
  packer.AddField("lastprice", "16.8");
  packer.EndPack();
  IT2UnPacker unpacker = this.t2.GetT2UnPacker(packer.PackerBuffer);
  // 发布消息
  try
  {
      /*
       * 主题发布时传入的过滤条件必须要和主题过滤条件完全匹配
       * 消息中心 2.0 相关资料详见《消息中心 2.0 开发 API 使用指南.doc》
       * 更多主题发布接口参见接口定义...
       */
      this.MC2Publisher.Publish(topicName, filter, packer.PackerBuffer);
  }
  catch (MC2Exception ex)
  {
      MessageUtility.Show(string.Format("主题: {0} 发布失败\r\nerrorNo: {1},
errorInfo: {2}",
          topicName, ex.ErrorNo, ex.ErrorInfo));
  }
  finally
  {
      // 释放回收打包解包器等资源
      this.t2.ReleaseT2UnPacker(unpacker);
      this.t2.ReleaseT2Packer(packer);
      filter.Dispose();
  }
```

```
}
```

注意

消息发布时必须提供发布主题所有过滤条件信息，否则无法成功发布消息。

发布消息支持两种格式：业务打包格式的内容发布 和 非业务打包格式的内容发布，详见接口定义或《消息中心 2.0 开发 API 使用指南.doc》

8 客户端身份标示

客户端的身份标示，如常说的操作员或登录用户，其在某一个客户端中是全局唯一的。

目前框架已经提供一个 IIdentity 接口用于表示当前操作员或登录用户，接口声明如下：

```
// 该接口存在于 Hundsun.Framework.Communication.dll 中
/// <summary>
/// 用户标示接口
/// </summary>
public interface IIdentity
{
    // 获取或设置操作员数值 Id
    long OperId { get; set; }
    // 获取或设置操作员的编号
    string OperCode{ get; set; }
    // 获取或设置操作员的名称
    string OperName{ get; set; }
    // 获取或设置当前操作员密码
    string OperPwd{ get; set; }
    // 获取或设置当前的语言环境信息
    // CultureInfo.Name 值，如 zh-CN，en-US，zh，en 等等
    string Language{ get; set; }
}
```

特定的业务应用程序需要一个类来实现该接口，该实现类将包含这个特定应用程序操作员特有的属性信息（如包含会员号，VIP 等等），例如将其命名为 AppIdentity。当用户登录成功后，需要从服务端返回的消息中解析出相关属性，构造一个 AppIdentity 对象，并且对该对象的相应属性赋值，最后必须对 IDBFactory.Identity 属性赋值，以保证该类实例在客户端中的全局唯一性。

可以查看 Hundsun.Framework.Client 的 FormLogin 窗体的登录事件的后台代码：

```
//TODO:登录成功后设置全局用户信息
//这里是演示
AppIdentity  appIdentity = new AppIdentity ();
```

```
appIdentity.OperCode = txtUserID.Text;
appIdentity.OperPwd = txtPwd.Text;
appIdentity.OperName = "Demo 用户";
appIdentity.O4SpecialProperty1 = "App 的操作员的特定属性 1";
appIdentity.O4SpecialProperty1 = "App 的操作员的特定属性 2";
//设置全局用户信息
this.communication.DBFactory.Identity = appIdentity;
```

9 日志和异常

9.1 日志

9.1.1 概述

系统提供了通用的日志记录接口，在需要记录业务日志或者记录异常信息等操作时，可以通过日志接口来完成。各日志接口实例一般是通过**依赖注入**通讯组件 **ICommunication**，并使用组件提供的接口工厂来获取，如下所示：

```
            //获取日志接口
            ILogUtility logUtility =
this.Communication.DBFactory.GetLogUtility(HSLogType.Exception);
```

另外，插件基类(AddInPartBase)现在已默认依赖注入 ICommunication 对象，所以在派生的插件中可以直接获取到该通信实例，并由此取得日志接口，而且还提供了一个直接获取日志接口的方法 GetLogUtility，可以像下面这样获取日志接口：

```
            //获取日志接口
            ILogUtility logUtility = this.GetLogUtility(HSLogType.Exception);
```

首先根据日志类型来获得一个 **ILogUtility** 的日志对象；然后可以写入日志，包括日志的级别，记录日志的位置和日志的信息。

HSLogLevel 定义了不同的日志级别；**HSLogType** 枚举类定义了不同的日志类型，分别对应不同的日志类型。

日志被记录在根目录下的 **Log** 目录中，根据日志的不同类型（HSLogType）保存在不同的文件夹中。

9.1.2 日志写入接口

```
        /// <summary>
        /// 写日志
        /// 传入用户信息为空则使用全局用户信息
        /// </summary>
```

```
        /// <param name="level">日志级别</param>
        /// <param name="location">记录日志的位置</param>
        /// <param name="message">日志信息</param>
        void WriteLog(HSLogLevel logLevel, string location, string message);
        /// <summary>
        /// 写日志
        /// </summary>
        /// <param name="level">日志级别</param>
        /// <param name="location">记录日志的位置</param>
        /// <param name="message">日志信息</param>
        /// <param name="userID">用户信息</param>
        voidWriteLog(HSLogLevellogLevel, stringlocation, stringmessage, string
userID);
```

9.1.3 日志类型

HSRCF 提供以下 6 种日志类型，各类型对应的日志文件保存在平台运行时执行目录下的 Log 子文件夹中，文件以当天日期为前缀、以日志类型为后缀。

例如，XXX\Log\SYS\2012-09-17_sys.log

业务开发人员一般只会用到 Exception，Business 和 Common 类型的日志。

```
    /// <summary>
    /// 日志类型
    /// </summary>
    public enum HSLogType
    {
        // 系统日志
        System = 0,
        // 通讯日志
        Communication = 1,
        // 自动更新日志>
        Upgrade = 2,
        // 异常
        Exception = 3,
        // 业务日志
        Business = 4,
        // 一般日志
        Common = 5
    }
```

9.1.4 日志级别

日志级别描述了当前记录信息的重要程度，各日志类型均有一个**日志记录级别值**，当且仅当

待写入的日志条目级别不小于该级别值时，才会被写入到日志文件中。

```
/// <summary>
/// 日志级别
/// 值越大级别越高，级别越高错误越严重
/// </summary>
public enum HSLogLevel
{
    // 调试
    DEBUG = 0,
    //信息
    INFO = 1,
    //警告
    WARN = 2,
    //错误
    ERROR = 3,
    //异常
    EXCEPTION = 4
}
```

9.2 异常

对于业务应用未捕获的异常，HSRCF 将显示异常的消息，并记录到日志，存于平台运行时执行目录下的 Log 子文件夹。具体的捕获方式，请参见 Hundsun.Framework.Platform 项目的 ShellApplication.cs 源文件。

业务需要捕获的框架异常，主要是通信方面的异常，如下表所示。

异常名	描述
T2Exception	T2 异常类的基类
T2CommunicationException	T2 通信异常
T2EsbMessageException	T2 报文异常类
T2PackerException	T2 打包异常类
T2UnPackerException	T2 解包异常类
MC2Exception	消息中心 2.0 通信异常

9.2.1 通信异常示例

```
try
{
DataTable LoginDt = t2.GetDataTable(new List<string>()
{ "account_content", "password" }, new List<object>() { txtUserID.Text,
this.txtPwd.Text }, functionId, 5000, null);
    //以下根据后台返回包内容正常取值
    CfaeIdentity identity = new CfaeIdentity();
    identity.AccountName = Convert.ToString(LoginDt.Rows[0]["account_name"]);
    identity.Account = Convert.ToString(LoginDt.Rows[0]["account"]);
    identity.PartCode = Convert.ToString(LoginDt.Rows[0]["part_code"]);
    identity.clientID = Convert.ToString(LoginDt.Rows[0]["client_id"]);
    identity.Language = Thread.CurrentThread.CurrentCulture.Name;
    this.communication.DBFactory.Identity = identity;
    this.Hide();
    .....
}
catch (T2CommunicationException ex)
{
    MessageBox.Show(ex.ErrorNo + ":" + ex.ErrorInfo);
//这里业务可以根据错误信息自定义处理
}
```

10 控件

10.1 概述

平台控件库除了实现标准控件之外，还实现了带标签系列的控件、多列下拉控件以及可折叠的面板等。基本上，控件库内的控件都实现了皮肤样式的处理。

控件库内的主要控件如下表所示。

分类	控件名	描述
组合控件	hsLabelTextBox	带标签的文本控件，实现正则匹配
	hsLabelComboBox	带标签的下拉控件，实现过滤、匹配等
	hsLabelNumericEditor	带标签的数值控件
	hsLabelDateTimePicker	带标签的日期时间控件，可以选择日期或时间类型
	hsLabelRangeDateTimePicker	带标签的有效日期或时间控件，可以选择日期或时间类型
	hsLabelDomainUpDown	带标签的文本选择控件
	hsLabelNumericUpDown	带标签的数值选择控件
	hsLabelTrackerBar	带标签跟踪条控件
	hsCheckBoxTextBox	带复选框和文本框控件
按钮控件	hsButton	标准按钮控件，其实现了自动校验功能
表格/列表控件	hsDataGridView	数据网格控件，实现了对 DataTable 类型数据的过滤、页脚汇总等
	hsCheckedListBox	复选框列表控件
	hsListBox	列表框控件

分类	控件名	描述
布局控件	hsPanel	标准容器控件，通过 DoLayout 方法实现动态布局
	hsGroupBox	带标题的容器控件
	hsTableLayoutPanel	表格形式排列的控件
	hsCollapsePanel	可折叠的容器面板，需要设置 Dock 属性和 ShowExpandIcon 才能显示可折叠标示，才能进行折叠/收缩
导航面板控件	hsXPanderPanelList	可折叠的面板控件
分隔形式控件	hsSplitContainerEx	单击分隔隐藏，大小可调的两个面板的组合面板控件
	hsSplitter	Dock 分隔控件
	hsSeparator	内容分隔控件
分页控件	hsPager	分页控件
树控件	hsTreeView	标准树控件
	hsThreeTreeView	三态树控件
卡片形式控件	hsTabControlEx	扩展的卡片控件
栏形式控件	hsToolStrip	工具栏
	hsStatusStrip	状态栏
	hsMenuStrip	菜单栏
其他	hsToolTip	提示控件
	hsContextMenuStrip	上下文菜单
	hsRadioButton	单选框
	hsCheckBox	复选框
	hsComboBox	下拉框，实现过滤、匹配等
	hsDomainUpDown	文本选择控件
	hsNumericUpDown	数值选择控件
	hsPictureBox	图片呈现控件
	hsProgressBar	进度条控件
	hsTrackBar	跟踪条控件
	hsDateTimePicker	日期时间选择器

分类	控件名	描述
	hsMonthCalendar	月历控件
	hsTextBox	文本控件
	hsScrollingText	显示滚动文本框
	hsRichTextBox	富文本框
	hsNumericEditor	数值框
	hsMaskedTextBox	掩码文本框

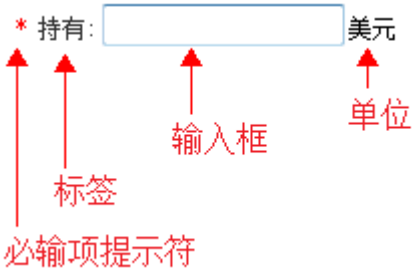
本节主要介绍表格控件(hsDataGridView)、标签系列组合控件和布局容器 hsPanel 的使用。关于这些控件所包括的接口信息，请参考**控件的 chm 帮助文档**。

10.2 标签系列组合控件

HSRCF 提供的带标签的组合控件如下表所示。

分类	控件名	描述
标签系列组合控件	hsLabelTextBox	带标签的文本控件，实现正则匹配
	hsLabelComboBox	带标签的下拉控件，实现过滤，匹配等
	hsLabelNumericEditor	带标签的数值控件
	hsLabelDateTimePicker	带标签的日期时间控件，可以选择日期或时间类型
	hsLabelRangeDateTimePicker	带标签的有效日期时间控件，可以选择日期或时间类型
	hsLabelDomainUpDown	带标签的文本选择控件
	hsLabelNumericUpDown	带标签的数值选择控件
	hsLabelTrackerBar	带标签跟踪条控件

它们通常由左右区域组成，左侧是标签和必输提示符（为红色星号*），右侧是输入框（文本，数值，日期时间等等）和单位。例如，下图所示控件为带标签的文本框(hsLabelTextBox)。



其他的组合控件布局类似，下面将对各控件进行详述。

10.2.1 hsLabelTextBox

概述

该控件是带标签的普通文本输入组合控件。

常用属性

属性名称	描述
Value	获取或设置一个值，该值为文本框中的文本值。
AllowEmpty	获取或设置一个值，该值指示值是否允许为空，它提供给界面校验使用。
LabelText	获取或设置一个值，该值指示标签的文本。
Connectors	获取或设置一个值，该值指示标签和输入框之间的连接符，默认冒号。
RegularMode	文本数据正则校验模式，如校验文本为 URL，Email，中国邮编，中国身份证等等，默认为 None。
CustomRegExp	获取或设置一个值，该值指示正则表达式，只当校验模式为 Custom 时，无须首位加入^$，系统会默认加入。
CustomErrorMessage	获取或设置一个值，该值指示正则校验错误时的提示信息，只当校验模式为 Custom 时。
CustomTheme	获取或设置一个值，该值指示是否启用系统主题样式，默认启用。
Enable	获取或设置一个值，该值指示控件是否可用，默认启用。
EnableLockMode	获取或设置一个值，该值指示是否启用锁定解锁模式，如果启用将在控件的最右侧出现一个锁的图片用于控制控件的 Enable 属性。

属性名称	描述
EnableSkipCyclicValidate	获取或设置一个值，该值指示当控件在循环校验列队中时候，是否跳过循环校验。
IsSkipNextControl	获取或设置一个值，该值指示是否跳转到上或下一个控件，默认是启用跳转。
LabelUnitBackColor	获取或设置一个值，该值指示单位的背景色。
LabelUnitForeColor	获取或设置一个值，该值指示单位的字体颜色。
MaxLength	获取或设置一个值，该值指示值的最大字节长度，默认为 32767，它提供给界面校验使用。
MinLength	获取或设置一个值，该值指示值的最小字节长度，默认为 0，它提供给界面校验使用。
TextBoxUnit	获取或设置一个值，该值指示单位。
TextBoxWidth	获取或设置一个值，该值指示输入区域的宽度。

常用事件

事件名称	描述
ValueChanged	当 Value 属性值更改时发生
HSValidating	用户自定义校验事件
HSValidated	校验成功后事件

常用方法

方法名称	描述
Focus()	控件设置输入焦点，如果输入焦点请求成功，为 true；否则为 false。
Clear()	启用循环即时校验时，清空控件值并且重置有效性。
Reset	启用循环即时校验时，重置有效性，使有效校验时能重新触发 HSValidated 事件。

开发步骤

1.) 从 Visual Studio 工具箱中拖入  hsLabelTextBox 放到界面上。

2.) 如果需要设置属性，则在界面上选择该控件，鼠标右键选择属性，打开属性设置窗口。

3.) 如果需要绑定控件的事件，打开属性设置窗口，选择事件分类  ，双击要绑定的事件名，Visual Studio 会生成事件处理方法。

取值赋值

取值

```
string myValue = this.hsLabelTextBox1.Value;
```

赋值

```
this.hsLabelTextBox1.Value = "13988888888";
```

10.2.2 hsLabelComboBox

下拉控件有 hsLabelComboBox 和 hsComboBox，其中 hsLabelComboBox 是带 Label 标签的组合控件，hsComboBox 是不带 Label 标签单独使用的。两种控件的使用方式和代码基本一样，提供的接口也是一致的，下面主要介绍 hsLabelComboBox 的使用方法。

常用属性

下表所示内容为 hsLabelComboBox 的常用属性，均放在设计视图的数据选项卡中。

属性名称	描述
AllowEmpty	是否允许空值，默认为 true(允许)。请看校验相关章节。
ComboBoxBackColor	用户自定义控件默认状态的背景色。（Enable 为 fasle 时不生效；仅在 CustomTheme 为 false，ReadOnly 也为 false 时有效）
ComboBoxForeColor	用户自定义控件默认状态的前景色（Enable 为 fasle 时不生效；仅在 CustomTheme 为 false，ReadOnly 也为 false 时有效）
ComboBoxFont	获取或设置 ComboBox 的字体。
ComboBoxSelectionModel	设置下拉控件是否为单选（Single）或者多选（Multiple）。
ComboBoxDropDown	设置下拉模式，有以下选项： ● DropDown：可以输入可以选 ● DropDownList：不能输入只能选
ComboBoxWidth	设置 ComboBox 的宽度。
Connectors	获取或设置 Label 和 ComboBox 的连接符。

属性名称	描述
CustomTheme	是否启用系统主题样式： ● True：使用框架默认样式 ● False：使用用户自定义样式
DefaultSelect	如果 ComboBox 数据源有数据，这里可以设置在加载数据源后默认选中第几项数据，-1 代表无选中项。该属性仅在给数据源 DataSource 赋值时起作用。
DisabledComboBoxBackColor	用户自定义控件不可用状态的背景色。（Enable 为 fasle 时不生效；仅在 CustomTheme 为 false，ReadOnly 也为 false 时有效）
DisabledComboBoxForeColor	用户自定义控件不可用状态的前景色。（Enable 为 fasle 时不生效；仅在 CustomTheme 为 false，ReadOnly 也为 false 时有效）
DisplayTextJoinStr	多选时，选中值之间采用指定连接符号进行拼接，目前只能是“;”或者“,”。
Enabled	是否禁止使用。（如果为 false 代表完全不能操作）
EnableSkipCyclicValidate	当控件在循环校验列队中时候，是否跳过循环校验，默认是不能 false（不能跳过校验）。
Includes	以“,”隔开的包含项列表。 （1.）如果未设置排除项，那么只显示包含项 （2.）如果设置了排除项，那么是显示包含项和排除项的差集。 （3.）如果未设置包含项，只设置排除项，那么是从原始数据源中排除掉相关数据
Excludes	以',隔开的排除项列表。
InsertNullItem	单选下拉列表添加一个空选择项，如果是 true 则新增一个空项在第一行。
Items	在设计时设置 ComboBox 的数据源，每行数据都以 ItemsSplitChar 指定的符号分隔列数据。以 Items 设置的数据源会自动生成 DataTable 数据源，字段个数会以第一行的分隔列数目生成，字段名从第一列开始以“0”，“1”，“n”来命名生成。 以 Items 设置的数据源，所有接口同 DataTable 数据源的接口。
ItemsSplitChar	当单独使用 Items 作为数据源时，每行数据项之间的分隔符号。

属性名称	描述
KeyName	用户设置数据源中主键字段的名称，一般设置为可见列第一列的字段名（或属性名）即可，下拉列表使用该字段名来过滤、匹配数据。 该值在设计时默认是不用设置的，运行时会自动取可见列第一列的字段名。（推荐） 如果在设计时设置了属性值，请保证该字段名一定存在于数据源中，否则会抛出异常。
LabelBackColor	获取或设置 Label 的背景色。
LabelForeColor	获取或设置 Label 的字体颜色。
LabelFont	获取或设置 Label 的字体。
LabelText	获取或设置 Label 的文本，设置后会自动加上 Connectors 连接符。
ReadOnly	是否只读，如果只读将只能通过程序赋值，不能通过鼠标、键盘等方式操作赋值。
ShowBottom	是否显示下拉框底部区域，多选时可用。
TagEx	获取或设置 TagEx 的值，可用于保存控件相关的一些用户数据
VisibleColumnIndexs	下拉列表要包含显示的数据项索引列表，以','分隔，例如（0，1）代表只显示数据源中的第 0 列和第 1 列，其他列则隐藏。

常用事件

事件名称	解释
HSDisplayingText	当下拉列表有且只有选中一项数据时，用户可以自定义显示到 Text 中的文本。选中多选数据是控件默认显示方式，和这个事件无关。该事件必须在给 DataSource 赋值前绑定
HSInitCustomStyle	用户自定义每个列表项的样式
HSValidating	校验事件（参见校验相关章节）
HSValidated	校验成功后事件（参见校验相关章节）
ValueChanged	用户自定义选中值改变事件（当有效选中状态发生改变时才会引发该事件）

常用方法

方法名称	描述
List<int> GetSelectItemIndexs()	返回选中项索引(返回值始终是数据源中的索引顺序)，多选时使用。
int GetSelectItemIndex()	返回选中项索引，-1 代表没有选中项(返回值始终是数据源中的索引顺序)，单选时使用。
List<T> GetSelectItems<T>()	多选取值，T 为数据源中数据项的数据类型。
T GetSelectItem<T>()	单选取值，T 为数据源中数据项的数据类型。
void SetValue<T>(List<T> list)	泛型赋值，list 是选中项列表，T 为赋值可转换类型。
List<T> GetValue<T>()	泛型取值，T 是取值时想要直接转换的数据类型。
List<T> GetValue<T>(int fieldIndex)	取指定字段值，fieldIndex 是指指定字段或者属性的索引值。
List<T> GetValue<T>(string fieldName)	取指定字段值：这里的 fieldName 是指指定字段或者属性的名称。
string GetValueToString(int fieldIndex)	取指定字段值，并且使用连接符连接，这里的 fieldIndex 是指指定字段或者属性的索引值。
string GetValueToString(string fieldName)	取指定字段值，并且使用连接符连接，这里的 fieldName 是指指定字段或者属性的名称。

开发步骤

1.) 拖入一个下拉控件到插件或者窗体中。

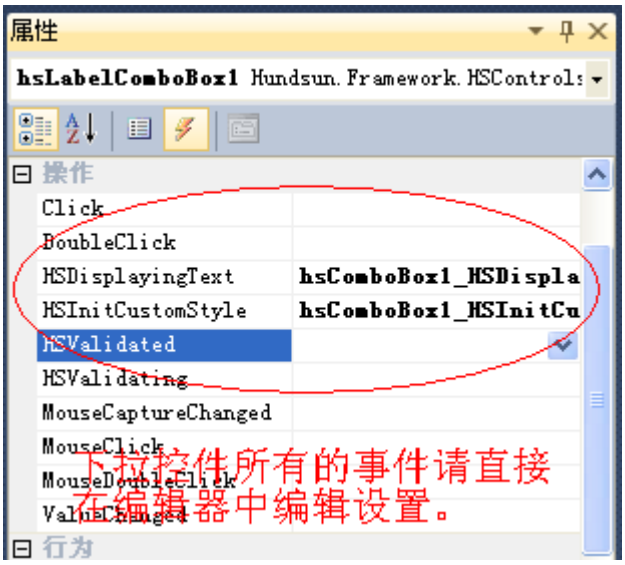
2.) 在设计视图设置控件的 Items 属性作为数据源或者直接用代码设置数据源。

如果是代码设置下拉控件的数据源，可以使用两种数据源格式，分别是泛型集合对象或者是 DataTable 对象。

例如：this.hsComboBox1.DataSource = 泛型集合或 DataTable 对象；

3.) 设置相关事件（可选）

下拉控件的部分事件属于绑定前事件，目前规定开发时只能在设计视图事件窗口添加所有相关事件。



示例：

```
string hsComboBox1_HSDisplayingText(object data)
{
//数据源如果是 DataTable 请将 data 强转成 DataRow， 数据源如果是泛型集合请将 data 强转成泛型对象
DataRow dr = data as DataRow;
 // 这里请根据要转换的对象进行相关操作
   return dr[this.hsComboBox1.KeyName].ToString();
}

void hsComboBox1_HSInitCustomStyle(object data， ListViewItem item)
{
//数据源如果是 DataTable 请将 data 强转成 DataRow， 数据源如果是泛型集合请将 data 强转成泛型对象
DataRow dr = data as DataRow;
// 这里请根据要转换的对象进行相关操作
if (row == null)
return;
    if (row["ID"].ToString().Contains("1"))
    {
        item.ForeColor = System.Drawing.Color.Red;
     }
    if (row["ID"].ToString().Contains("2"))
    {
        item.ForeColor = System.Drawing.Color.Yellow;
     }
     if (row["ID"].ToString().Contains("3"))
     {
```

```
        item.ForeColor = System.Drawing.Color.Green;
    }
}
```

取值/赋值

取值方式：

```
string s = this.hsLabelComboBox1.Value;//单多选取值
string s0 = this.hsLabelComboBox1.GetValue<string>()[0];//多选
String s1 = this.hsLabelComboBox1.GetValueToString(fieldName);//取指定字段值
GetSelectItem()方法：用于返回选中项关联的对象
```

取值注意相关的判断：

```
var value = this. hsLabelComboBox1.GetValue<string> ();
if (value!= null && value.Count > 0)
{
    …
}
```

赋值方式：

```
this.hsLabelComboBox1.Value = "值";//单多选赋值
this.hsLabelComboBox1.SetValue(new List<string>() { "值 1", "值 2", "值 3" });
//多选赋值
```

全键盘操作

键（组合键）		单选		多选	
		无下拉	**有下拉**	**无下拉**	**有下拉**
Escape			关闭下拉		关闭下拉
Alt			关闭下拉		关闭下拉
Alt	Up Down PageDown PageUp	显示下拉		显示下拉	
Up		上一焦点控件	上一条选项	上一焦点控件	上一条选项
Down		下一焦点控件	下一条选项	下一焦点控件	下一条选项

PageUp			上一页选项（保持 ListView 的默认翻页效果）		上一页选项
PageDown			下一页选项		下一页选项
Left		上一条选项			
Right		下一条选项			
Space					切换当前选项的状态（选中/不选）
Enter		下一焦点控件	关闭下拉框	下一焦点控件	关闭下拉框
Ctrl	A				全选
	Z				全部取消

10.2.3 hsLabelNumericEditor 和 hsLabelNumericUpDown

概述

这两个控件是带标签的数值输入组合控件。其中 hsLabelNumericUpDown 是带微调按钮的，它们的接口基本一致，除非特别说明。下面将对它们一起说明。

常用属性

属性名称	描述
Value	获取或设置一个值，该值指示数值框上的数值。
ChineseNumberAlign	获取或设置一个值，该值指示中文样式的金额的显示位置，默认是控件底部下。
ChineseNumberOffset	获取或设置一个值，该值指示大写金额提示信息显示的位置偏移。
DataRangeType	获取或设置一个值，该值指示控件的数值范围类型，P 代表 Positive，即正的，如 NumericRangeType.PInt32，代表正整数；N 代表 Negative，即负的，如 NumericRangeType.NInt32，代表负整数。其他类推。
DecimalPlaces	获取或设置要显示的十进制小数位数，默认值为 0，只有在数值范围设定为的 Float，PFloat，NFloat，Decimal，PDecimal，Decimal，Any 时候才能设置小数位数。
LabelText	获取或设置一个值，该值指示标签的文本。

属性名称	描述
Connectors	获取或设置一个值，该值指示标签和输入框之间的的连接符，默认冒号。
CustomTheme	获取或设置一个值,该值指示是否启用系统主题样式,默认启用。
Enable	获取或设置一个值，该值指示控件是否可用，默认启用。
EnableLockMode	获取或设置一个值，该值指示是否启用锁定解锁模式，如果启用将在控件的最右侧出现一个锁的图片用于控制控件的 Enable 属性。
EnableSkipCyclicValidate	获取或设置一个值，该值指示当控件在循环校验列队中是，是否跳过循环校验。
IsSkipNextControl	获取或设置一个值，该值指示是否跳转到上或下一个控件，默认是启用跳转。
LabelUnitBackColor	获取或设置一个值，该值指示单位的背景色。
LabelUnitForeColor	获取或设置一个值，该值指示单位的字体颜色。
MaxAllow	获取或设置一个值，该值指示值是否允许到达最小值，默认值是 true。
MinAllow	获取或设置一个值，该值指示值是否允许到达最大值，默认值是 false。
Maximum	最大值，默认值是 79228162514264337593543950335。指定的数值范围的最大值不同，如 PInt16 最大值为 32767，NInt16 最大值为 0；PInt32 最大值为 2147483647，NInt32 最大值为 0；其他类推。
Minimum	最小值，默认值是-79228162514264337593543950335。只有指定的数值范围的最小值不同，如 PInt16 最小值为 0，NInt16 最小值为-32768；PInt32 最小值为 0；NInt32 最小值为-2147483648；其他类推。
hsLabelNumericEditor 的 NumeriEditorUnit 属性 hsLabelNumericUpDown 的 NumericUpDownUnit 属性	获取或设置一个值，该值指示单位。
hsLabelNumericEditor 的 NumeriEditorWidth 属性 hsLabelNumericUpDown 的 NumericUpDownWidth 属性	获取或设置一个值，该值指示输入区域的宽度。

属性名称	描述
ShowChineseNumber	获取或设置一个值，该值指示是否提示中文样式的金额，默认值不显示。
ThousandsSeparator	获取或设置一个值，该值指示是否显示千位分隔符，默认值为 false。

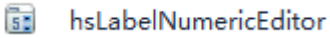
常用事件

事件名称	描述
ValueChanged	当 Value 属性值更改时发生
HSValidating	用户自定义校验事件
HSValidated	校验成功后事件

常用方法

方法名称	描述
Focus()	为控件设置输入焦点，如果输入焦点请求成功，该方法返回值为 true；否则为 false。
Clear()	启用循环即时校验时，清空控件值并且重置有效性。
Reset	启用循环即时校验时，重置有效性，使有效校验时能重新触发 HSValidated 事件。

开发步骤

1.) 从 Visual Studio 工具箱中拖入 

或者

放到界面上。

2.) 如果需要设置属性，则在界面上选择该控件，鼠标右键选择属性，打开属性设置窗口。

3.) 如果需要绑定控件的事件，打开属性设置窗口，选择事件分类  ，双击要绑定的事件名，Visual Studio 会生成事件处理方法。

取值赋值

取值

```
decimal myEditorValue = this.hsLabelNumericEditor.Value;
```

```
decimal myUpDownValue = this.hsLabelNumericUpDown.Value;
```

赋值

```
this. hsLabelNumericEditor.Value = 500;
this.hsLabelNumericUpDown.Value = 500;
```

10.2.4 hsLabelDateTimePicker

概述

该控件表示带标签的日期时间组合控件，它能选择只显示日期或时间，或显示日期时间，以及自定义显示格式。

常用属性

属性名称	描述
Value	获取或设置一个值，该值指示控件的日期时间值。
ShowCheckBox	获取或设置一个值，该值指示在选定日期的左侧是否显示一个复选框。如果在选定日期的左边显示复选框，则为 true；否则为 false。默认值为 false。
Checked	获取或设置一个值，该值指示是否已用有效日期/时间值设置了 System.Windows.Forms.DateTimePicker.Value 属性且显示的值可以更新。
ShowUpDown	获取或设置一个值，该值指示是否使用数值调节钮控件（也称为 up-down 控件）调整日期/时间值。如果数值调节钮控件用于调整日期/时间值，则为 true；否则为 false。默认值为 false。
Format	获取或设置一个值，该值指示控件中显示的日期和时间格式。
CustomFormat	获取或设置一个值，该值指示自定义日期/时间格式字符串。
LabelText	获取或设置一个值，该值指示标签的文本。
Connectors	获取或设置一个值，该值指示标签和输入框之间的连接符，默认冒号。
CustomTheme	获取或设置一个值，该值指示是否启用系统主题样式，默认启用。
Enable	获取或设置一个值，该值指示控件是否可用，默认启用。
EnableLockMode	获取或设置一个值，该值指示是否启用锁定解锁模式，如果启用将在控件的最右侧出现一个锁的图片用于控制控件的 Enable 属性。

属性名称	描述
EnableSkipCyclicValidate	获取或设置一个值，该值指示当控件在循环校验列队中时候，是否跳过循环校验。
IsSkipNextControl	获取或设置一个值，该值指示是否跳转到上或下一个控件，默认是启用跳转。
LabelUnitBackColor	获取或设置一个值，该值指示单位的背景色。
LabelUnitForeColor	获取或设置一个值，该值指示单位的字体颜色。
MaxDate	获取或设置一个值，该值指示值可在控件中选择的最大日期和时间，默认最大值为 12/31/9998 00:00:00。
MinDate	获取或设置一个值，该值指示值可在控件中选择的最小日期和时间，默认最小值为 1/1/1753 00:00:00。
DateTimePickerWidth	获取或设置一个值，该值指示输入区域的宽度。

常用事件

事件名称	描述
ValueChanged	当 Value 属性值更改时触发
HSValidating	用户自定义校验事件
HSValidated	校验成功后事件
CloseUp	当下拉日历被关闭并消失时触发
DropDown	当显示下拉日历时发生

常用方法

方法名称	描述
Focus()	为控件设置输入焦点，如果输入焦点请求成功，则为 true；否则为 false。
Clear()	启用循环即时校验时，清空控件值并且重置有效性。
Reset	启用循环即时校验时，重置有效性，使有效校验时候能重新触发 HSValidated 事件。

开发步骤

1.) 从 Visual Studio 工具箱中拖动  hsLabelDateTimePicker 至界面上。

2.) 如果需要设置属性，则在界面上选择该控件，鼠标右键选择属性，打开属性设置窗口。

3.) 如果需要绑定控件的事件，打开属性设置窗口，选择事件分类  ，双击要绑定的事件名，Visual Studio 会生成事件处理方法。

取值赋值

取值

```
DateTime myValue = this.hsLabelDateTimePicker1.Value;
```

赋值

```
this. hsLabelDateTimePicker1.Value = DateTime.Now;
```

10.2.5 hsLabelRangeDateTimePicker

概述

该控件表示带标签的日期范围（如开始日期，结束日期）或时间（如开始时间，结束时间）范围的组合控件。

常用属性

属性名称	描述
BeginValue	获取或设置一个值，该值指示开始的 DateTimePicker 的值，该值需要小于等于 EndValue，而且需要在最大值，最小值之间。
EndValue	获取或设置一个值，该值指示结束的 DateTimePicker 的值，该值需要大于等于 BeginValue，而且需要在最大值，最小值之间。
MaxDate	获取或设置一个值，该值指示值可在控件中选择的最大日期和时间，默认最大值为 12/31/9998 00:00:00。
MinDate	获取或设置一个值，该值指示值可在控件中选择的最小日期和时间，默认最小值为 1/1/1753 00:00:00。
FormatType	获取或设置控件中显示的日期和时间格式类型，默认显示日期范围。
LabelText	获取或设置一个值，该值指示标签的文本。
Connectors	获取或设置一个值，该值指示标签和输入框之间的连接符，默认冒号。
CustomTheme	获取或设置一个值，该值指示是否启用系统主题样式，默认启用。
Enable	获取或设置一个值，该值指示控件是否可用，默认启用。

属性名称	描述
EnableLockMode	获取或设置一个值，该值指示是否启用锁定解锁模式，如果启用将在控件的最右侧出现一个锁的图片用于控制控件的 Enable 属性。
EnableSkipCyclicValidate	获取或设置一个值，该值指示当控件在循环校验列队中时候，是否跳过循环校验。
IsSkipNextControl	获取或设置一个值，该值指示是否跳转到上或下一个控件，默认是启用跳转。
LabelUnitBackColor	获取或设置一个值，该值指示单位的背景色。
LabelUnitForeColor	获取或设置一个值，该值指示单位的字体颜色。

常用事件

事件名称	描述
BeginValueChanged	当开始 DateTimePicker 的日期时间的 Value 属性值更改时发生。
EndValueChanged	当结束 DateTimePicker 的日期时间的 Value 属性值更改时发生。
HSValidating	用户自定义校验事件
HSValidated	校验成功后事件

常用方法

方法名称	描述
Focus()	为控件设置输入焦点，如果输入焦点请求成功，则为 true；否则为 false。
Clear()	启用循环即时校验时，清空控件值并且重置有效性。
Reset	启用循环即时校验时，重置有效性，使有效校验时能重新触发 HSValidated 事件。

开发步骤

1.) 从 Visual Studio 工具箱中拖入  hsLabelRangeDateTimePicker 放到界面上。

2.) 如果需要设置属性，则在界面上选择该控件，鼠标右键选择属性，打开属性设置窗口。

3.) 如果需要绑定控件的事件，打开属性设置窗口，选择事件分类  ，双击要绑定的事件名，Visual Studio 会生成事件处理方法。

取值赋值

取值

```
DateTime beginValue = this. hsLabelRangeDateTimePicker1.BeginValue;
DateTime endValue = this. hsLabelRangeDateTimePicker1.EndValue;
```

赋值

```
    this. hsLabelRangeDateTimePicker1.BeginValue=DateTime.Now;
    this. hsLabelRangeDateTimePicker1.EndValue=DateTime.Now;
```

10.2.6 hsLabelTrackerBar

概述

该控件是带标签的跟踪条组合控件。

常用属性

属性名称	描述
Value	获取或设置一个值，该值跟踪条上滚动框的当前位置的数值。
LabelText	获取或设置一个值，该值指示标签的文本。
Connectors	获取或设置一个值，该值指示标签和输入框之间的连接符，默认冒号。
CustomTheme	获取或设置一个值，该值指示是否启用系统主题样式，默认启用。
Enable	获取或设置一个值，该值指示控件是否可用，默认启用。
EnableLockMode	获取或设置一个值，该值指示是否启用锁定解锁模式，如果启用将在控件的最右侧出现一个锁的图片用于控制控件的 Enable 属性。
EnableSkipCyclicValidate	获取或设置一个值，该值指示当控件在循环校验列队中时，是否跳过循环校验。
IsSkipNextControl	获取或设置一个值，该值指示是否跳转到上或下一个控件，默认是启用跳转。
LabelUnitBackColor	获取或设置一个值，该值指示单位的背景色。
LabelUnitForeColor	获取或设置一个值，该值指示单位的字体颜色。
Maximum	获取或设置一个值，该值指示此 TrackBar 使用的范围的上限。
Minimum	获取或设置一个值，该值指示此 TrackBar 使用的范围的下限。
TickFrequency	获取或设置一个值，该值指定控件上绘制的刻度之间的增量。

属性名称	描述
TickStyle	获取或设置一个值，该值指示如何显示跟踪条上的刻度线。
TrackBarUnit	获取或设置一个值，该值指示单位。
TrackBarWidth	获取或设置一个值，该值指示输入区域的宽度。

常用事件

事件名称	描述
ValueChanged	当 Value 属性值更改时发生
HSValidating	用户自定义校验事件
HSValidated	校验成功后事件

常用方法

方法名称	描述
Focus()	为控件设置输入焦点，如果输入焦点请求成功，则为 true；否则为 false。
Clear()	启用循环即时校验时，清空控件值并且重置有效性。
Reset	启用循环即时校验时，重置有效性，使有效校验时候能重新引发 HSValidated 事件。

开发步骤

1.) 从 Visual Studio 工具箱中拖入  hsLabelTrackerBar 放到界面上。

2.) 如果需要设置属性，则在界面上选择该控件，鼠标右键选择属性，打开属性设置窗口。

3.) 如果需要绑定控件的事件，打开属性设置窗口，选择事件分类  ，双击要绑定的事件名，Visual Studio 会生成事件处理方法。

取值赋值

取值

```
int myValue = this.hsLabelTrackerBar1.Value;
```

赋值

```
this. hsLabelTrackerBar1.Value = 10;
```

10.2.7 hsLabelDomainUpDown

概述

该控件是带标签的字符串滚动显示框。

常用属性

属性名称	描述
Value	获取或设置一个值，该值为控件的文本值。
DomainUpDownItems	获取该控件对象集合。
LabelText	获取或设置一个值，该值指示标签的文本。
Connectors	获取或设置一个值，该值指示标签和输入框之间的连接符，默认冒号。
CustomTheme	获取或设置一个值,该值指示是否启用系统主题样式,默认启用。
Enable	获取或设置一个值，该值指示控件是否可用，默认启用。
EnableLockMode	获取或设置一个值，该值指示是否启用锁定解锁模式，如果启用将在控件的最右侧出现一个锁的图片用于控制控件的 Enable 属性。
EnableSkipCyclicValidate	获取或设置一个值，该值指示当控件在循环校验列队中时候，是否跳过循环校验。
IsSkipNextControl	获取或设置一个值，该值指示是否跳转到上或下一个控件，默认是启用跳转。
LabelUnitBackColor	获取或设置一个值，该值指示单位的背景色。
LabelUnitForeColor	获取或设置一个值，该值指示单位的字体颜色。
DomainUpDownUnit	获取或设置一个值，该值指示单位。
DomainUpDownWidth	获取或设置一个值，该值指示输入区域的宽度。

常用事件

事件名称	描述
SelectedItemChanged	在更改 SelectedItem 属性后触发
HSValidating	用户自定义校验事件
HSValidated	校验成功后事件

常用方法

方法名称	描述
Focus()	为控件设置输入焦点，如果输入焦点请求成功，则为 true；否则为 false。
Clear()	启用循环即时校验时，清空控件值并且重置有效性。
Reset	启用循环即时校验时，重置有效性，使有效校验时能触发引发 HSValidated 事件。

开发步骤

1.) 从 Visual Studio 工具箱中拖入  hsLabelDomainUpDown 放到界面上。

2.) 如果需要设置属性，则在界面上选择该控件，鼠标右键选择属性，打开属性设置窗口。

3.) 如果需要绑定控件的事件，打开属性设置窗口，选择事件分类  ，双击要绑定的事件名，Visual Studio 会生成事件处理方法。

取值赋值

取值

```
string myValue = this.hsLabelDomainUpDown1.Value;
```

赋值

```
this. hsLabelDomainUpDown1.Value = "item1";
```

11 界面布局

11.1 表格控件的列布局

11.1.1 概述

恒生系列控件中包括了表格控件 **hsDataGridView**。因此开发过程中需要对表格控件的布局进行设置。

HSRCF 提供动态布局设置，主要包括列的布局设置如下所示：

1.) 列标题

2.) 列显示顺序

3.) 列可见性

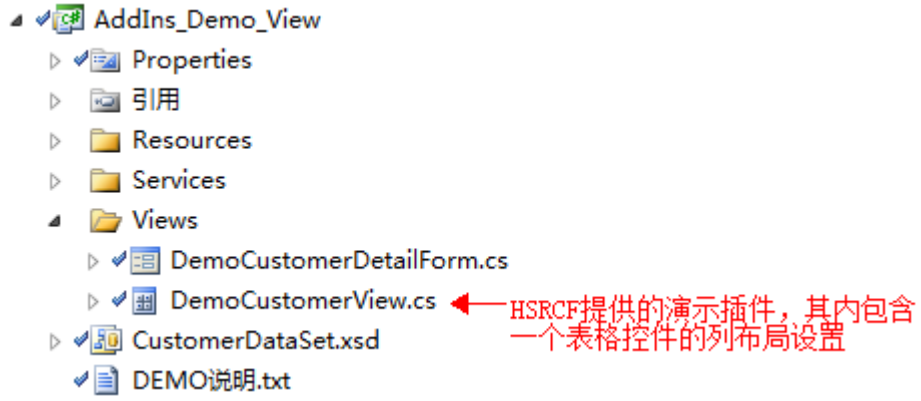
4.) 列可冻结性

hsDataGridView 本身已经提供了AutoSizeColumnsMode来自动调整列标题的宽度和单元格的宽度。

11.1.2 开发步骤

演示项目

打开 HSRCF 的解决方案，其内包含如下演示工程项目：



其中 DemoCustomerView 为插件。其中表格控件期望呈现的运行时界面如下所示。

姓名	编号	地址	电话	手机	邮箱	级别	类型	时间	金额
Name 1000	1000	浙江	0571-28820616	16666662256	cd@163.com	VIP	客户类型2	2013/7/30 9:51	943
Name 999	999	浙江温州	0571-28820368	16666668906	ef@yahoo.com	Common	客户类型	2013/4/22 9:51	93400
Name 998	998	浙江嘉兴	0571-28820740	16666665456	ab@hundsun.com	Super	客户类型3	2013/4/23 9:51	19305
Name 997	997	浙江杭州	0571-28820482	16666665976	someone@hund...	VIP	客户类型2	2013/4/24 9:51	22148
Name 996	996	浙江宁波	0571-28820601	16666660936	ab@hundsun.com	Common	客户类型	2013/4/25 9:51	94090

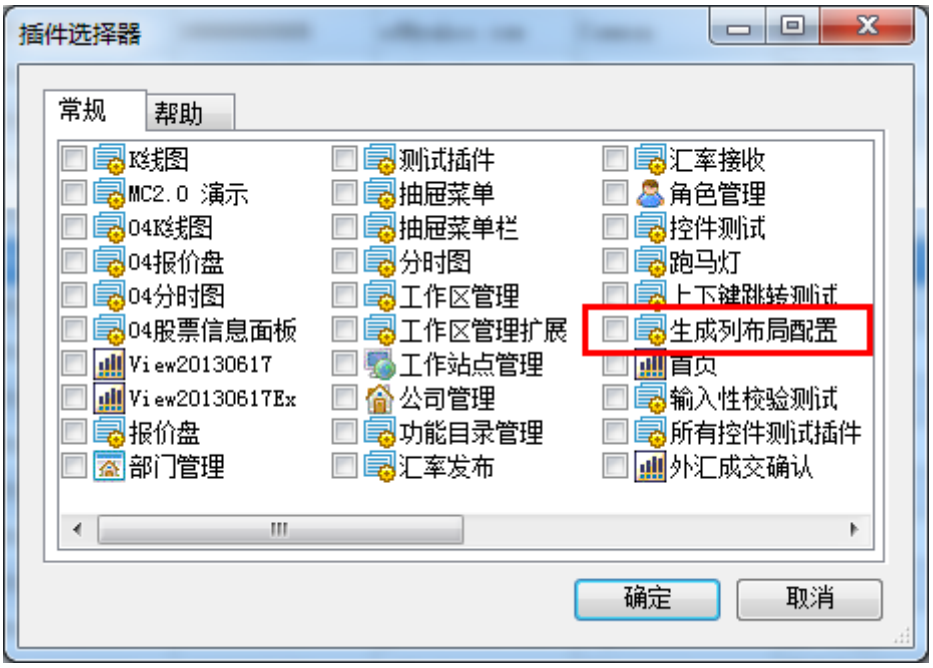
要完成该表格控件的列布局设置需要几个步骤：

步骤1 生成列布局配置

HSRCF 提供了两种方法生成列布局配置信息，一种是通过**生成列布局设置**插件，一种是通过**列布局配置**按钮。

1.) 通过工具插件

HSRCF 提供了简单**生成列布局设置**插件供开发人员生成列布局信息，登录统一客户端开发平台，添加如下方框中标示的插件。

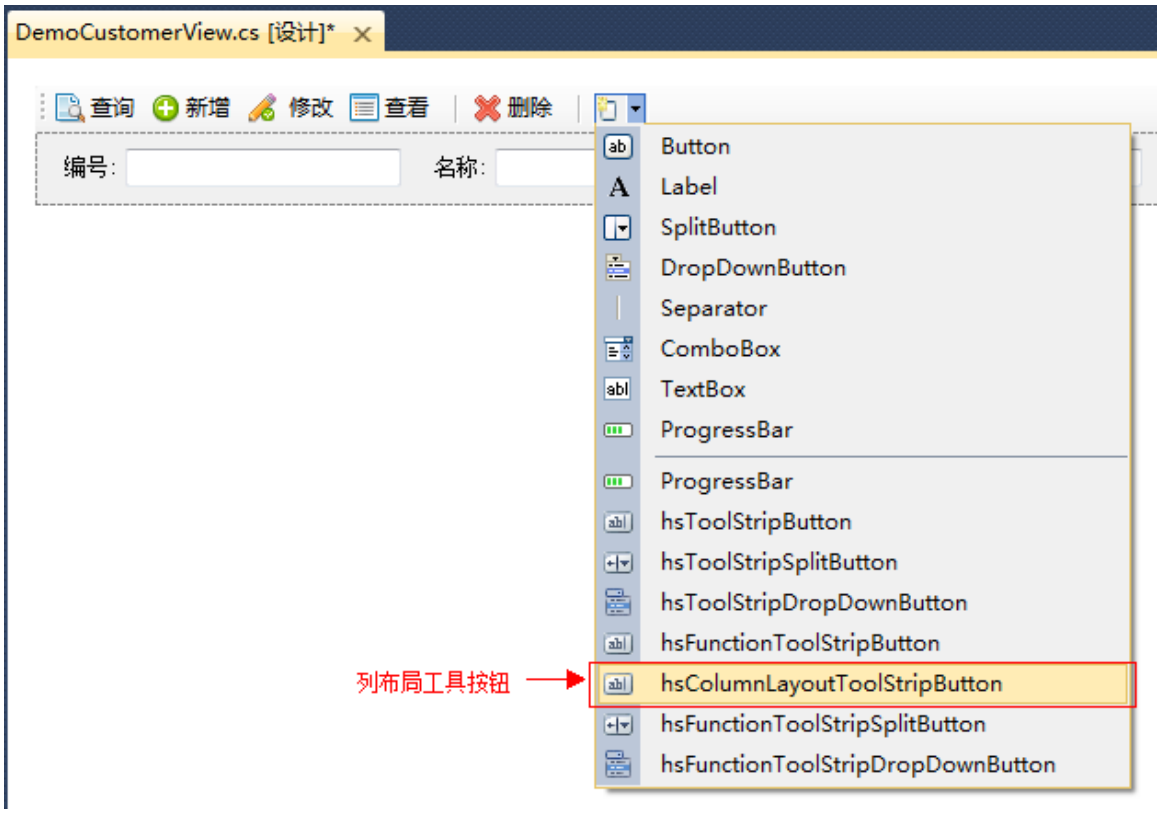


单击“确定”后，得到如下界面。

- **填入功能号**，一般填入查询业务的功能号即可，该功能号用于分组，例如填入 5。
- **填入网格控件名称**，即该表格控件的 Name 属性值，该名称用于区分同一功能号下不同表格控件的布局信息。例如填入 hsDataGridView1
- **填入列名和列显示标题**，这两个大文本控件内，每一行数据代表一列的列名和列标题。例如填入列名称：Name，CustomerID，Addr，OfficeTel，Phone，Email，Level，Type，TradeDate，TradeMoney，对应列显示标题：名称，编号，地址，电话，手机，邮箱，级别，类型，时间，金额。
- **单击“生成列布局”按钮**，将在右侧的表格控件中生成列布局信息，以及在最右侧的列配置结果(Xml)中生成最终配置结果(XML 形式)。可以改变右侧的表格控件中生成列布局信息，它将自动反应到配置结果(XML)中。
- 在编辑完所有信息后，单击“复制到剪贴板”按钮将配置结果复制到剪贴板中，可以将剪贴板中的数据放在一个临时文本文件中待用。

2.) 通过列布局按钮

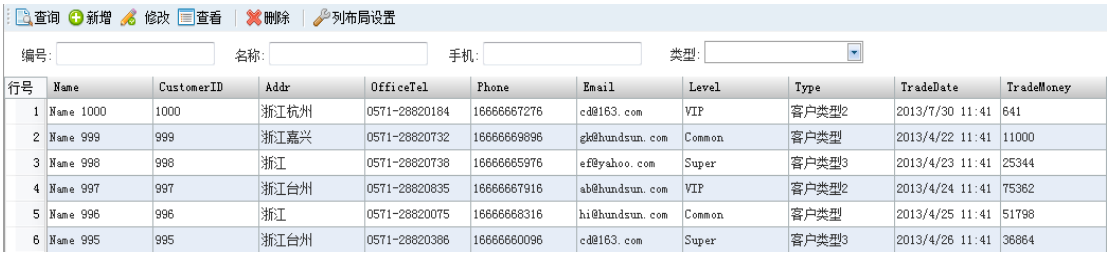
当编写完表格控件的数据绑定功能（即查询得到的数据能绑定到表格控件）后，通常可以在插件的工具栏中增加列布局按钮，如下图所示。



然后设置该按钮的以下属性：

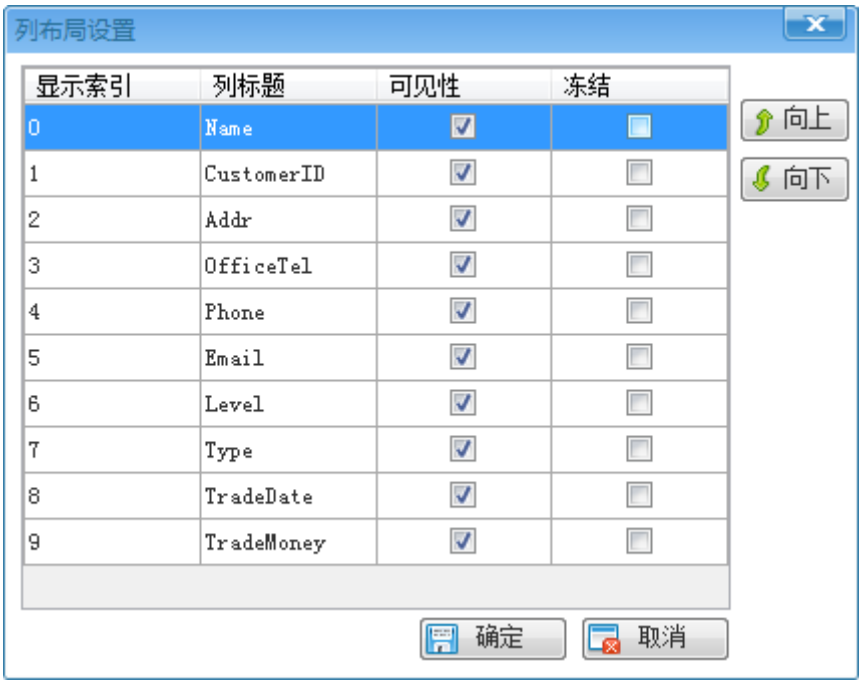
属性	描述
FunctionID	功能号，目前使用该功能号来分组。
TargetGrid	需要进行配置的表格控件（必须是 hsDataGridView 类型）。
CommandType	显示的按钮图片，建议使用 Tool。
DisplayStyle	显示的方式，建议选择 ImageAndText。

- 运行统一客户端开发平台，打开演示的插件，单击“查询”按钮（数据将绑定到表格控件上），如下图所示。

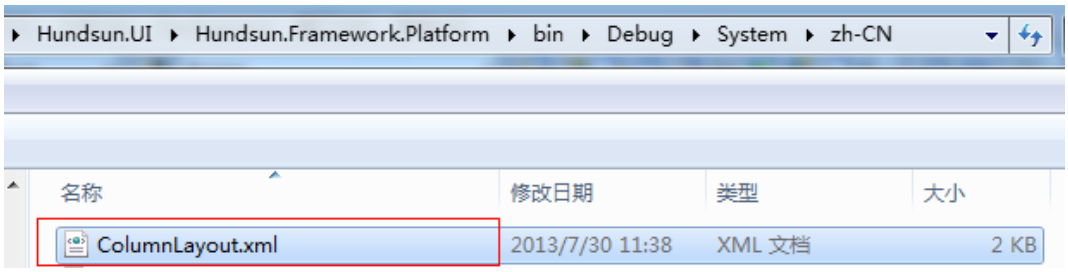


行号	Name	CustomerID	Addr	OfficeTel	Phone	Email	Level	Type	TradeDate	TradeMoney
1	Name 1000	1000	浙江杭州	0571-28820184	16666667276	cd@163.com	VIP	客户类型2	2013/7/30 11:41	641
2	Name 999	999	浙江嘉兴	0571-28820732	16666669896	gk@hundsun.com	Common	客户类型	2013/4/22 11:41	11000
3	Name 998	998	浙江	0571-28820738	16666665976	ef@yahoo.com	Super	客户类型3	2013/4/23 11:41	25344
4	Name 997	997	浙江台州	0571-28820835	16666667916	ab@hundsun.com	VIP	客户类型2	2013/4/24 11:41	75362
5	Name 996	996	浙江	0571-28820075	16666668316	hi@hundsun.com	Common	客户类型	2013/4/25 11:41	51798
6	Name 995	995	浙江台州	0571-28820386	16666660096	cd@163.com	Super	客户类型3	2013/4/26 11:41	36864

- 单击列布局配置按钮，此时平台弹出列布局设置窗口，如下图所示。



可以修改显示索引、列标题、可见性和冻结列等，修改完后，单击“确定”后系统将保存列布局配置信息到运行时执行目录下的 System\zh-CN(en)\ColumnLayout.xml 文件，如下图所示。

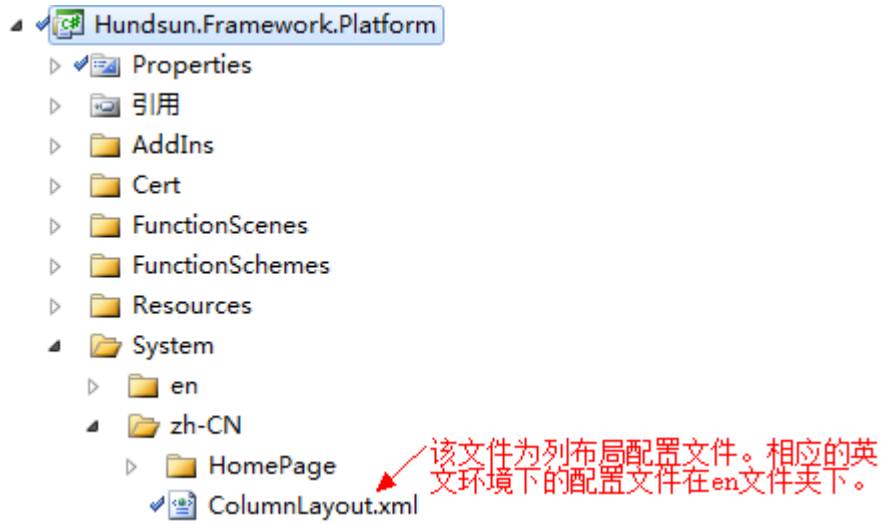


打开该配置文件，提取如下配置信息到一个临时的文本文件待用。

```
<Group Name="5">
  <Grid Name="hsDataGridView1">
    <Column Name="Name" DisplayIndex="0" HeaderText="姓名" Visible="True" Frozen="False" Width="128" />
    <Column Name="CustomerID" DisplayIndex="1" HeaderText="编号" Visible="True" Frozen="False" Width="108" />
    <Column Name="Addr" DisplayIndex="2" HeaderText="地址" Visible="True" Frozen="False" Width="107" />
    <Column Name="OfficeTel" DisplayIndex="3" HeaderText="电话" Visible="True" Frozen="False" Width="163" />
    <Column Name="Phone" DisplayIndex="4" HeaderText="手机" Visible="True" Frozen="False" Width="135" />
    <Column Name="Email" DisplayIndex="5" HeaderText="邮箱" Visible="True" Frozen="False" Width="172" />
    <Column Name="Level" DisplayIndex="6" HeaderText="级别" Visible="True" Frozen="False" Width="138" />
    <Column Name="Type" DisplayIndex="7" HeaderText="类型" Visible="True" Frozen="False" Width="104" />
    <Column Name="TradeDate" DisplayIndex="8" HeaderText="时间" Visible="True" Frozen="False" Width="130" />
    <Column Name="TradeMoney" DisplayIndex="9" HeaderText="金额" Visible="True" Frozen="False" Width="124" />
  </Grid>
</Group>
```

步骤2 修改列布局配置文件

打开列布局配置文件 ColumnLayout.Xml。



将临时文本中的配置结果信息，拷贝到该 Xml 文件的<ColumnLayout>节点下，如下所示：

```
<?xml version="1.0" standalone="yes"?>
<ColumnLayout>
  <Group Name="5">
    <Grid Name="hsDataGridView1">
      <Column Name="Name" DisplayIndex="0" HeaderText="姓名" Visible="True" Frozen="False" Width="128" />
      <Column Name="CustomerID" DisplayIndex="1" HeaderText="编号" Visible="True" Frozen="False" Width="108" />
      <Column Name="Addr" DisplayIndex="2" HeaderText="地址" Visible="True" Frozen="False" Width="107" />
      <Column Name="OfficeTel" DisplayIndex="3" HeaderText="电话" Visible="True" Frozen="False" Width="163" />
      <Column Name="Phone" DisplayIndex="4" HeaderText="手机" Visible="True" Frozen="False" Width="135" />
      <Column Name="Email" DisplayIndex="5" HeaderText="邮箱" Visible="True" Frozen="False" Width="172" />
      <Column Name="Level" DisplayIndex="6" HeaderText="级别" Visible="True" Frozen="False" Width="138" />
      <Column Name="Type" DisplayIndex="7" HeaderText="类型" Visible="True" Frozen="False" Width="104" />
      <Column Name="TradeDate" DisplayIndex="8" HeaderText="时间" Visible="True" Frozen="False" Width="130" />
      <Column Name="TradeMoney" DisplayIndex="9" HeaderText="金额" Visible="True" Frozen="False" Width="124" />
    </Grid>
  </Group>
</ColumnLayout>
```

步骤3 加载列布局配置

打开 DemoCustomerView 插件，在 OnLoad 方法或 Load 事件中加入如下语句：

```
        /// <summary>
        /// Load 事件
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void CustomerView_Load(object sender, EventArgs e)
        {
            //设置不自动产生列，该设置应该在已经配置了列布局信息后设置
            this.hsDataGridView1.AutoGenerateColumns = false;
            //加载列布局，不自动计算列宽，列宽的调整将根据表格控件的 AutoSizeColumnMode 属性进行自动调整
            //5 即填入的功能号，用于分组；hsDataGridView1 为需要布局的表格控件引用。
            ColumnLayoutUtility.LoadColumnLayout("5", this.hsDataGridView1);
        }
```

11.1.3 注意事项

在 HSRCF 系列控件中，列布局按钮有以下三个：

- 独立按钮控件：hsColumnLayoutButton 控件
- 工具项：hsColumnLayoutToolStripButton 工具项控件
- 菜单项：hsColumnLayoutToolStripMenuItem 菜单项控件

这三个控件在配置了相关属性（目标分组和目标表格控件）后，都能直接打开列布局配置窗体。如果用户需要个性化设置列布局，可以将这些按钮可见性置为可见即可。

11.2 标签系列控件的布局

11.2.1 概述

HSRCF 中布局是**运行时动态布局**，所有待布局的控件需要在 **hsPanel** 控件中，目前**不支持嵌套布局，**即如果当前 hsPanel1 内包含 hsPanel2，布局 hsPanel1 将不对 hsPanel2 内的控件进行布局。**所以在嵌套情况下，请布局最内层的 hsPanel,以保证布局还是可控的。**

如果运行时动态布局满足不了业务要求也可以采用手工布局方式，当前的 IDE(如 Visual Studio)都提供了很强大的设计布局支持。

而 hsPanel 布局很简单，只需要简单调用 DoLayout 方法即可完成布局。具体请看布局控件 hsPanel。

布局的基本流程如下：

1.) 收集目标容器（hsPanel）内待布局控件，将其分成两类，一类是**按钮类**控件（如 hsButton），一类**非按钮类**控件（如 hsLabelTextBox 等）。

2.) 然后将这两类控件按照 **Tab 索引值（该值即控件的 TabIndex 属性值）**从小到大排序待用；

3.) 对**非按钮类控件**进行横向或纵向布局。

4.) 对按钮类控件进行布局，按钮类控件布局在非按钮类控件之后，**最后一行**即放置按钮类控件使用。**按钮类控件右对齐**。

5.) 返回能容纳布局区域的大小。

因为 hsPanel 目前支持纵向和横向布局，下面将分别简述纵向和横向情况下的布局。

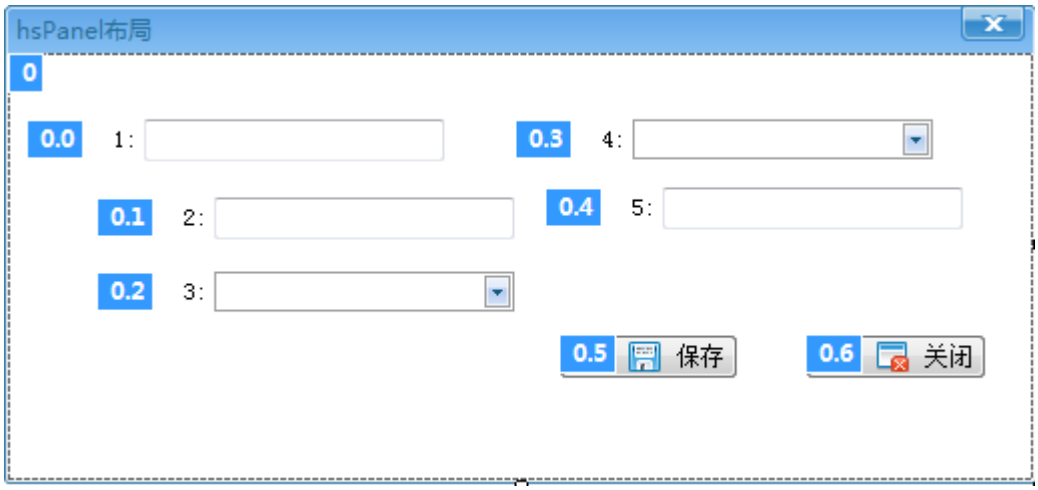
11.2.2 纵向布局

纵向布局是指非按钮类控件被按照一列一列布局，即布局完第一列后再布局第二列，然后布局第三列，以此类推。其中按钮类控件始终在最后一行靠右对齐布局。

纵向布局需要设置 hsPanel 的LayoutOrientation属性值为 **Vertical**。

hsPanel 默认布局成两列，可以设置**LayoutCol 属性**值来控制列数。

下图为设计期的纵向布局示意图。



0.0 是 Tab 索引值。可以通过 Visual Studio 的工具显示 Tab 键顺序的按钮来查看上图所有控件的 Tab 键索引值，工具按钮如下所示：

0.0 指的是它的父容器的索引值为 0，它自己的索引值为 0；其他类似。

上图中的 Tab 键的顺序为 0->0.0->0.1->0.2->0.3->0.4->05->0.6

运行期顺序如下图所示。

其中行列情况如下表所示：

	行列情况	
列	第一列	由控件 1，2，3 组成
	第二列	由控件 4，5 组成

	行列情况	
行	第一行	由控件 1，4 组成
	第二行	由控件 2，5 组成
	第三行	由控件 3 组成

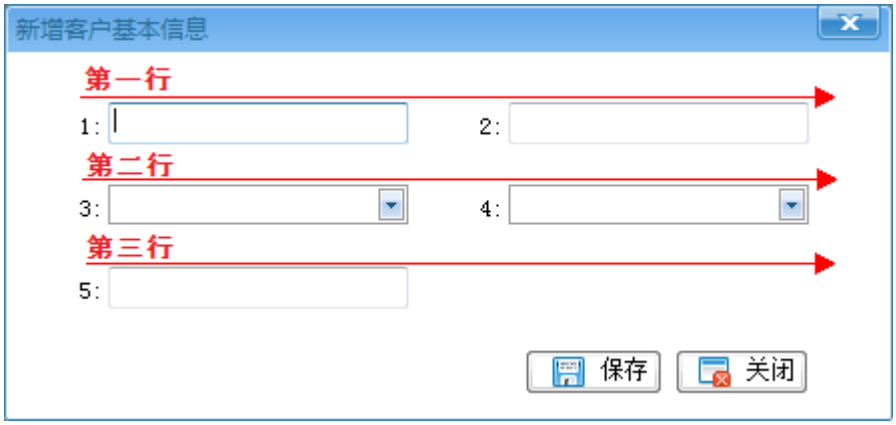
11.2.3 横向布局

横向布局是指非按钮类控件按照一行一行布局，即布局完第一行后再布局第二行，然后布局第三行，以此类推。其中按钮类控件始终在最后一行靠右对齐布局。

横向布局需要设置 hsPanel 的LayoutOrientation属性值为 **Horizontal**。

设计期示意图纵向布局的设计期示例图。

运行期效果图如下图所示。



其中行列情况如下表所示：

	行列情况	
列	第一列	由控件 1，3，5 组成
	第二列	由控件 2，4 组成
行	第一行	由控件 1，2 组成
	第二行	由控件 3，4 组成
	第三行	由控件 5 组成

11.2.4 布局控件 hsPanel

属性名称	描述
LayoutCol	Panel 内，控件排列后的列数，默认为 2 列。该属性与控件的排列方向无关
LayoutIntervalButton	获取或设置最下面一行各按钮之间的间隔
LayoutIntervalHorizontal	获取或设置控件列间隔（布局列距）
LayoutIntervalVertical	获取或设置控件行间隔（布局行距）
LayoutMargin	获取或设置容器内上下左右边距的距离（以像素为单位），默认为 15，15，20，20
LayoutOrientation	容器内布局排列方式。Horizontal：横向布局；Vertical：纵向布局

注意

属性设置完成以后，一定要在窗体或插件的 Load 事件中调用指定 hsPanel 的 **DoLayout()**方法！

```
/// <summary>
/// 布局管理
/// reLayout 表示是否刷新布局；True 表示刷新布局；false 表示不刷新；默认为 false
/// </summary>
/// <param name="reLayout">是否刷新布局；True 表示刷新布局；false 表示不刷新；默认为 false</param>
public Size DoLayout(bool reLayout = false)
```

11.2.5 其他容器控件的布局

其他容器控件，全部需要在内部最外层再放一个 hsPanel，然后再使用 hsPanel 布局。

11.2.6 注意事项

多 Tab 页布局问题

当将控件放入 Tab 页的 hsPanel 容器控件中时，如果想布局各个 Tab 页，并且避免在切换 Tab 页时布局 hsPanel 控件，可以在 Load 事件中写入如下代码，以便一次性的排序多个 Tab 页内的 hsPanel 容器控件。

```
            //保存原先选中的 Tab 页的索引
```

```
            int oldSelectIndex = this.hsTabControl1.SelectedIndex;
            //循环各个 Tab 页，设置每页类 hsPanel 的布局
            for (int i = 0; i < this.hsTabControl1.TabCount; i++)
            {
                this.hsTabControl1.SelectedIndex = i;
                if (this.hsTabControl1.TabPages[i].Controls.Count > 0)
                {
                    //获取 hsPanel 控件（在这里 hsPanel 控件是 Tab 页下的第一个控件）
                    //如果不是，可以采用循序遍历的方式找到想要的 hsPanel 控件
                    hsPanel pl = this.hsTabControl1.TabPages[i].Controls[0] as
hsPanel;
                    if (pl != null)
                    {
                        //布局 hsPanel
                        pl.DoLayout();
                    }
                }
            }
            //还原选中的 Tab 页的索引
            this.hsTabControl1.SelectedIndex = oldSelectIndex;
```

12 界面数据校验

12.1 概述

界面数据校验是指对界面上的控件值进行校验。除了控件内属性对控件值的约束外，还有一些自定义数据校验的逻辑。而校验成功后，需要做一些操作，如修改其他控件的值、锁定控件可用性等。HSRCF 提供的界面数据校验均提供这几个方面的功能。

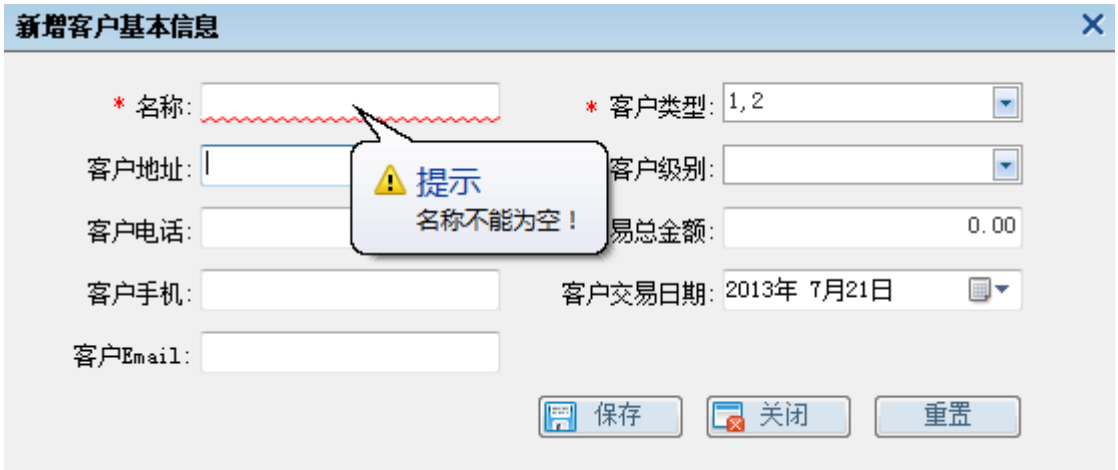
界面校验分为即时校验和统一校验两部分，它们完整构建了 HSRCF 校验体系。

12.1.1 即时校验

即时校验，即在界面操作时，非按钮类可输入性控件操作时触发的校验，它的校验是即时发生的，是发生在可输入性控件内的。

HSRCF 根据触发校验方式的不同将即时校验分为失去焦点的即时校验（又称单一即时校验）和得到焦点的即时校验（又称循环即时校验），它们确切的含义和如何触发校验，请查看后续章节。

示例如下图所示。



单一即时校验

失去焦点的即时校验，又称**单一即时校验**，它是指在一个界面上当前获得焦点的控件，失去焦点后，系统将引发即时校验。

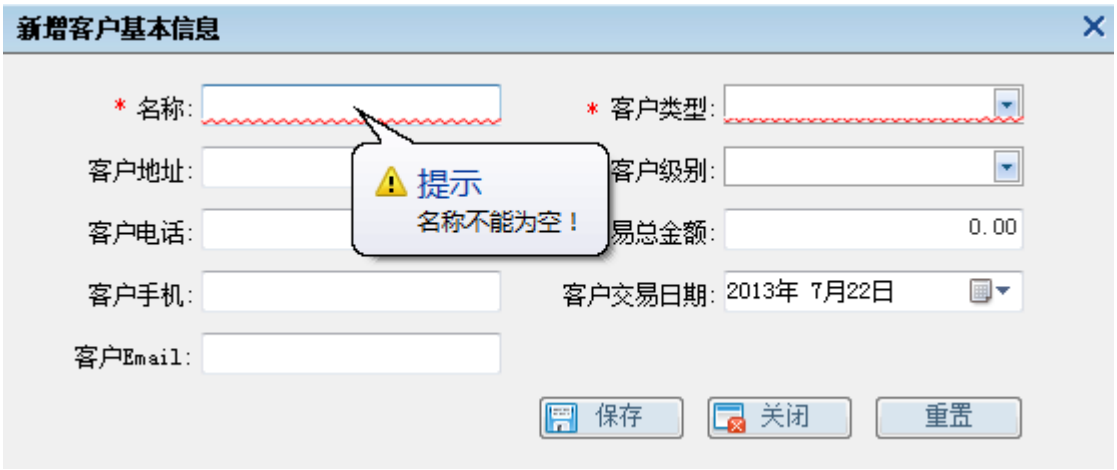
循环即时校验

得到焦点的即时校验，又称**循环即时校验**，它是指在一个界面上当某输入性控件获得焦点时将触发其他输入性控件的界面校验的即时校验。

12.1.2 统一校验

统一校验，即在界面操作时，按钮类控件操作触发的校验，它常常表现为一次性触发所有控件的校验，校验成功后通常做数据持久化操作，但是统一校验不显示针对控件的校验，因为它也包含逻辑校验。

按钮类控件本身不校验，而是触发其他输入性控件的校验。



如上图所示，当单击保存按钮时，名称校验和客户类型校验不通过，不能保存客户信息。

12.2 校验流程

12.2.1 即时校验流程

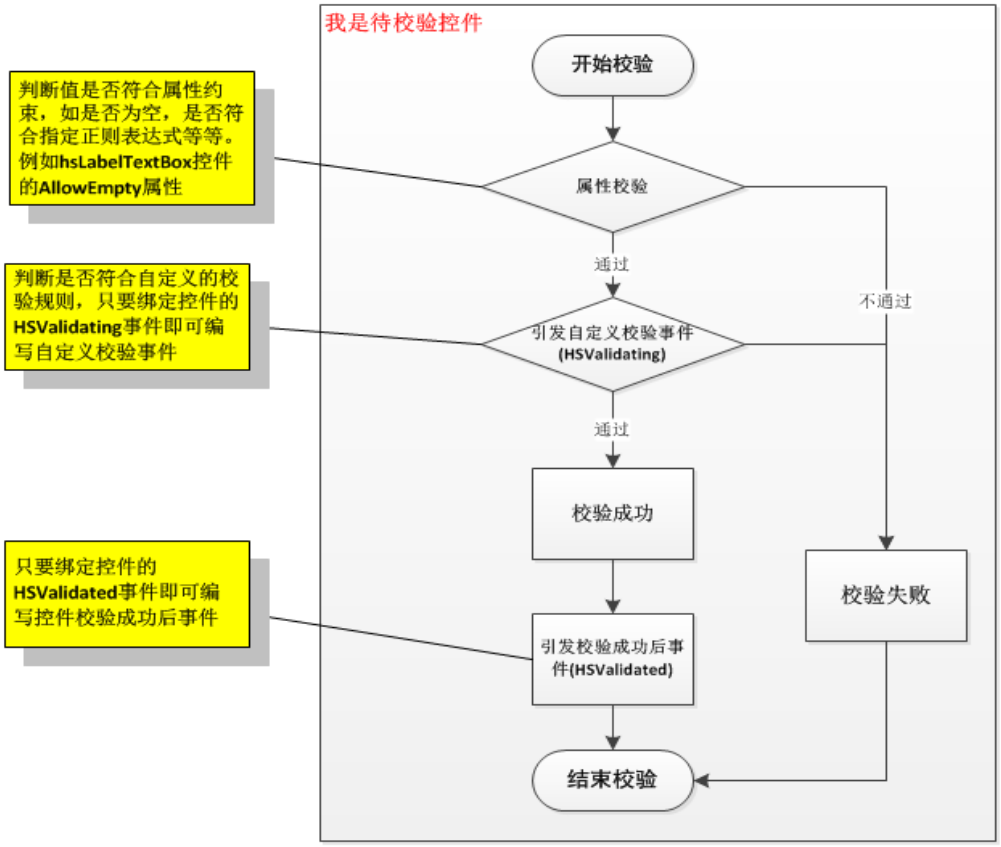
即时校验流程，主要需要关注的是即时校验是如何被触发的。下面将分不同章节讲解单一即时校验和循环即时校验的流程。

控件的内部校验流程

控件内需要判断校验部分包括如下：

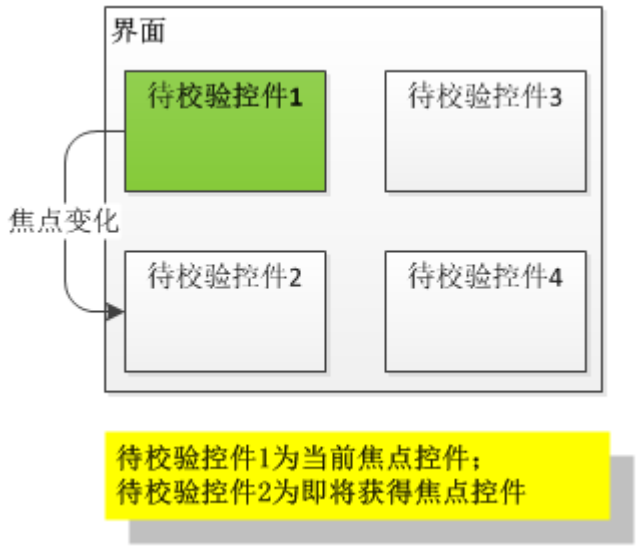
- 控件的属性的校验
- 控件的自定义校验事件
- 控件校验成功后事件

这几部分的执行顺序如下图所示。



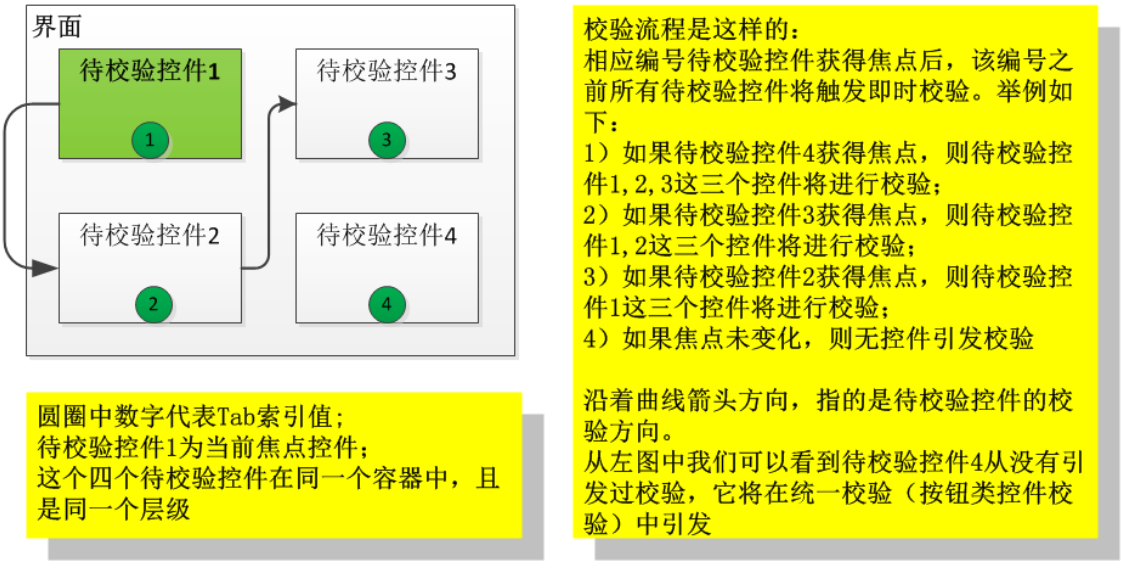
单一即时校验流程

假设界面上有四个待校验控件（待校验控件 1，待校验控件 2，待校验控件 3，待校验控件 4），待校验控件 1 为当前获得校验的控件，如下图所示。

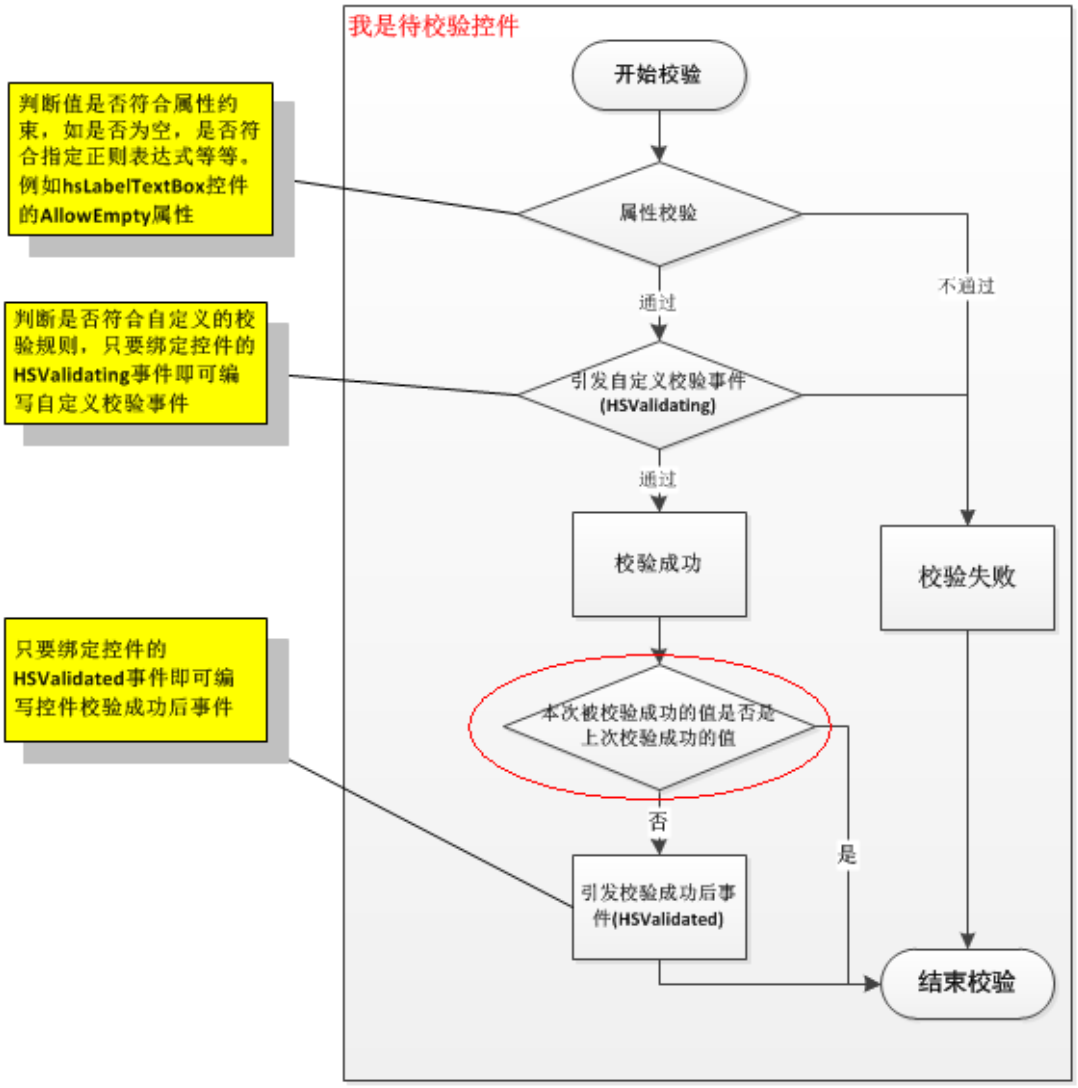


如果此时待校验控件 1 失去焦点（如鼠标单击待校验控件 2，按 Tab 键跳转等），则触发待校验控件 1 的单一即时校验。待校验控件 1 的内部校验流程正如控件内部校验流程讲述的方式执行。

循环即时校验流程



循环校验的待校验控件的内部校验流程和控件内部校验流程执行的方式有一点小区别，它的执行方式如下图所示，红色圆圈标示部分为不同点。



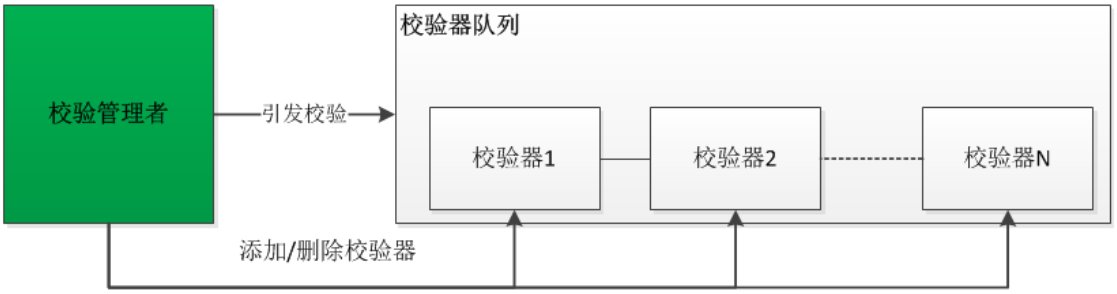
如果应用需要使用循环单一即时校验，建议使用 HSRCF 的 hsPanel 作为容器，并使用HSRCF提供的动态布局以达到待校验的控件按照 Tab 索引的顺序排列。

12.2.2 统一校验流程

统一校验不显示针对控件的校验，它也包含逻辑校验。统一校验的发起者一般都是按钮类组件，在 HSRCF 系列控件中，主要有以下三种类型：

- 独立按钮控件：hsButton 控件及其子类
- 工具项：hsToolStripButton 工具项及其子类
- 菜单项：hsToolStripMenuItem 菜单项及其子类

统一校验的流程其实并不复杂，其理念是有一个校验器队列用于保存界面校验器，还有一个校验管理者负责取出校验器队列中的校验器进行校验，如下图所示。



触发统一校验的控件一般是按钮类控件，而触发校验需要调用校验管理者的 DoValidate 方法来实现触发校验。

统一校验步骤如下：

1.) 通过校验管理者添加或删除校验器

2.) 通过调用校验管理器 DoValidate 方法来触发校验器队列中的校验器

校验器

校验器可以是任意的类型，其只要从 Hundsun.Framework.HSControls.UIValidater 类派生即可。通常不需要直接实现校验器，因为 HSRCF 已经提供了 CustomValidater，它是自定义校验器，能满足用户自定义校验规则要求。

校验器可以被分组和触发校验，UIValidater 简化声明如下所示。

```
    /// <summary>
    /// 表示界面校验器
    /// </summary>
    public abstract class UIValidater
    {
        /// <summary>
        /// 校验器校验
        /// </summary>
        /// <param name="errorMessage">错误消息（输出参数）</param>
        /// <param name="errorControls">校验错误的控件（输出参数）</param>
        /// <returns>返回是否校验成功</returns>
        public abstract bool Validate(out string errorMessage,  out Control[]
errorControls);

        /// <summary>
        /// 分组名称
        /// </summary>
        public string GroupName { get; set; }
    }
```

校验管理者

HSRCF 中默认的校验管理者是HSRCF 插件和HSRCF 窗体。它们都实现了统一校验管理者接口（Hundsun.Framework.HSControls.IValidate 接口），如下所示。

```
    /// <summary>
    /// 统一校验管理器接口
    /// </summary>
    public interface IValidate
    {
        /// <summary>
        /// 校验校验器队列中的校验器
        /// </summary>
        /// <returns>true 表示校验通过，false 表示校验不通过</returns>
        bool DoValidate();
        /// <summary>
        /// 分组校验校验器队列中的校验器
        /// </summary>
        /// <param name="GroupName">校验器所在的分组名</param>
        /// <returns>true 表示校验通过，false 表示校验不通过</returns>
        bool DoValidate(string GroupName);
        /// <summary>
        /// 删除指定分组的校验器
        /// </summary>
        /// <param name="groupName">指定分组名</param>
        void RemoveValidaters(string groupName);
        /// <summary>
        /// 移除 UI 校验器
        /// </summary>
        /// <param name="uiValidater">将被删除的校验器</param>
        void RemoveValidater(UIValidater uiValidater);
        /// <summary>
        /// 添加自定义规则的界面校验器
        /// </summary>
        /// <param name="validateExecuter">指定校验规则，该规则通过<see
cref="ExecuteValidate"/>委托来实现</param>
        UIValidater AddCustomValidator(ExecuteValidate validateExecuter);
    }
```

通过**校验管理者**接口，可以看到**校验管理者**能进行添加删除校验器、分组校验和全部校验等操作。

- 添加/删除校验器

  从校验管理者章节知道，HSRCF 提供的HSRCF 插件和HSRCF 窗体默认是校验管理者，即它们实现了校验管理者接口，可以通过它们添加和删除校验器。

- 触发校验

从校验管理者章节知道，要触发校验，只要调用校验管理器HSRCF 插件和HSRCF 窗体的***DoValidate()***方法即可。

- 触发分组校验

  从校验管理者章节知道，要触发校验，只要调用校验管理器HSRCF 插件和HSRCF 窗体的***DoValidate(string groupName)***方法即可，groupName 指的是分组名称。

12.3 自动校验功能(即时校验和统一校验的整合)

12.3.1 由来

首先，从统一校验管理者知道要触发统一校验，必须调用其 DoValidate 方法完成校验，为什么不能内置于按钮类控件中自动调用呢？

其次，单一即时校验能针对每个待校验控件完成校验触发，但是循环校验不能完成每个待校验控件的校验触发，因为最后一个 Tab 索引的待校验控件的校验无法触发，那怎么办？

再次，单一即时校验和循环即时校验都只是针对待校验控件值，当不判断控件值，如判断系统时间在指定时间后才生效等类似逻辑校验时，就束手无策了。

最后，对于即时校验一个应用系统中，用户只能选择一种即时校验启用，即要么选择单一即时校验，要么选择循环即时校验，可以对这种选择进行封装。

综合考虑以上几点，将即时校验和统一校验进行了整合，完成了**自动校验**的功能。

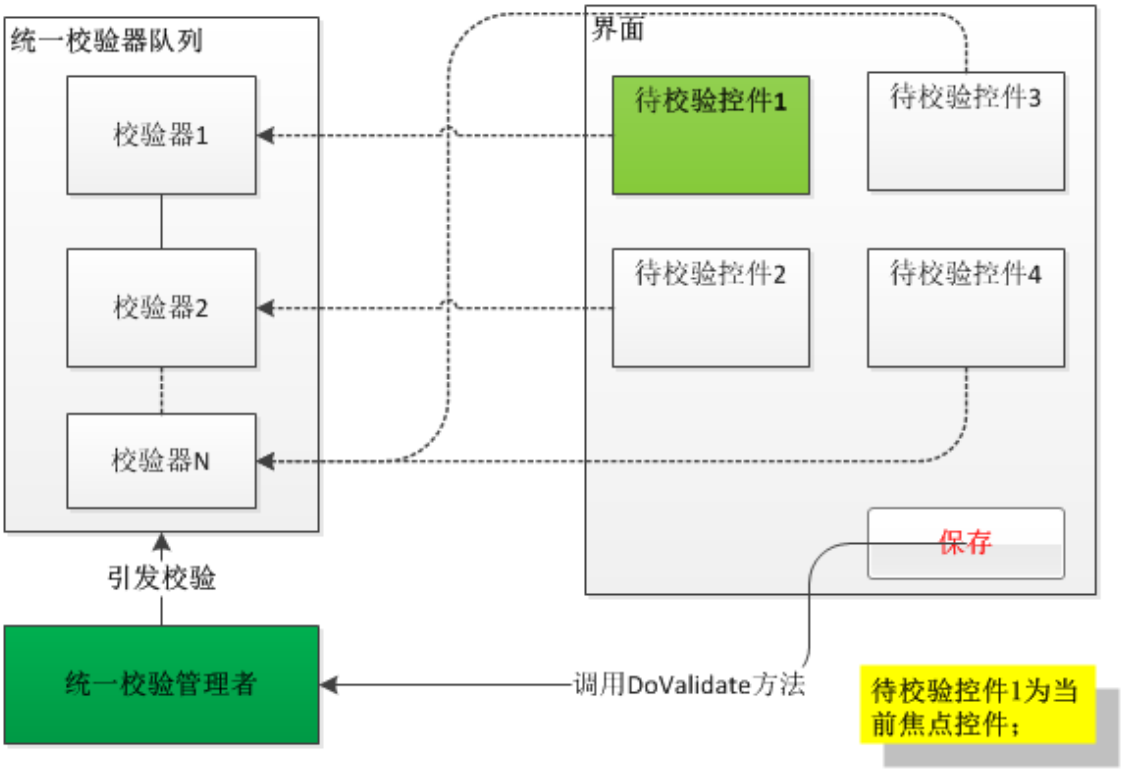
12.3.2 功能

自动校验主要完成以下功能：

- 智能选择即时校验类型，并将即时校验封装为统一校验的自定义校验器加入校验器队列。
- 自动调用统一校验管理者的 DoValidate 接口方法，免去了用户手动调用 DoValidate 方法的繁琐。

因为自动校验自动调用了 DoValidate 方法，其弥补了循环即时校验时最后一个 Tab 索引的待校验控件无法触发校验的问题。

12.3.3 自动校验流程



虚线代表待校验控件将映射一个统一校验的校验器；**实线**代表校验引发顺序，按钮=>统一校验管理者=>校验器队列=>校验器。

自动校验流程简述如下：

1.) 用户单击**保存按钮（或按钮类组件）**，保存按钮将收集当前目标校验容器中的待校验控件，并创建对应的统一校验自定义校验器。

2.) 获取按钮所处的统一校验管理者，并将创建的自定义校验器加入校验器队列中。

3.) 设置所有自定义校验器的分组为当前按钮分组。

4.) 通过调用 DoValidate(string groupName) 引发统一校验。

12.3.4 引发自动校验的按钮类控件

引发自动校验在 HSRCF 中是按钮类组件，在 HSRCF 系列控件中，主要有以下三种类型：

- 独立按钮控件：hsButton 控件及其子类
- 工具项：hsToolStripButton 工具项及其子类
- 菜单项：hsToolStripMenuItem 菜单项及其子类

12.4 结论

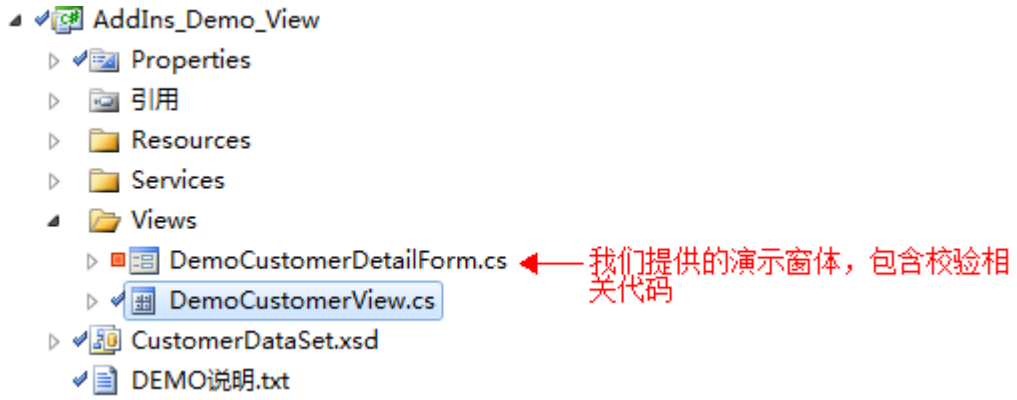
由于单独使用即时校验或者统一校验均无法满足校验功能，建议业务应用选择自动校验来完成界面校验的功能。

12.5 校验案例

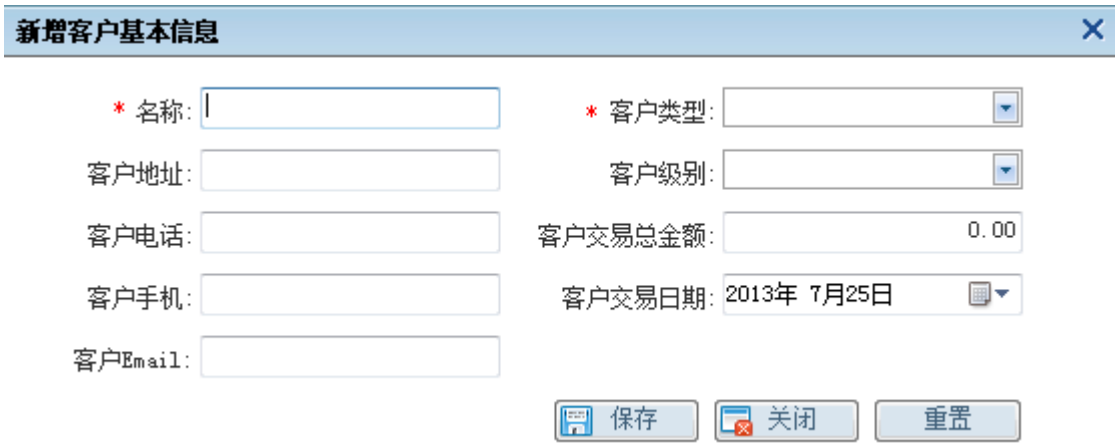
本节讲解如何使用自动校验和即时校验。

12.5.1 概述

打开 HSRCF 的解决方案，其包含的演示工程项目如下图所示。



其中 DemoCustomerDetailForm 为弹出窗体。期望呈现的界面如下图所示。



12.5.2 步骤

步骤1 即时校验

在设计完期望呈现的界面后，添加需要校验的属性，如下表所示。

属性	控件类型	需要的校验	控件内校验方式	具体方法
名称	hsLabelTextBox	文本值不能为空	属性校验	设置控件的 **AllowEmpty** 为 **false**
客户电话	hsLabelTextBox	需要符合规范：3 或 4 位区号-7 或 8 位电话	属性校验	设置控件的 **RegularMode** 属性为 **Tel**
客户手机	hsLabelTextBox	需要符合规范：11 位手机号	属性校验	设置控件的 **RegularMode** 属性为 **Phone**
客户 Email	hsLabelTextBox	需要符合规范：Email 格式	属性校验	设置控件的 **RegularMode** 属性为 **Email**
客户类型	hsLabelComboBox	文本值不能为空	属性校验	设置控件的 **AllowEmpty** 为 **false**
交易日期	hsLabelDateTimePicker	交易日期不能超过当前日期	自定义校验事件 HSValidating	绑定控件的 HSValidating 事件，编写校验逻辑

其中交易日期的自定义校验事件方法如下：

```
private void dtpTradeDate_HSValidating(object sender, HSValidatingEventArgs e)
        {
            //dtpTradeDate 为交易日期控件
            if (this.dtpTradeDate.Value.Date > DateTime.Today)
            {
                e.ErrorMessage = "交易日期不能大于今天";
                e.Result = false;
            }
        }
```

步骤2 按钮自动校验

期望的界面上有三个按钮，其中的保存按钮需要做界面校验。假设该保存按钮的控件名称为 **btnOK**，按钮的控件类型为 **hsButton**。

在 HSRCF 中能支持自动校验按钮类控件请参考引发自动校验的按钮类控件，它们都支持以下属性的设置。

1.) 需要设置如下按钮属性

按钮属性	描述	具体方法
EnableAutoValidate	该属性值表示按钮是否启用自动校验功能。true 表示启用按钮的自动校验功能;false 表示不启用按钮的自动校验功能。	设置该属性的值为 **true**。
TargetValidateContainer	该属性表示目标校验的容器，即将搜索待校验控件的范围。Null 表示将搜索范围限定为按钮的父容器；否则限定为设置的目标容器。待校验控件只要在该容器中即可，支持待校验控件放在目标容器的嵌套的容器中。	**不设置即可**。因为按钮的父容器和待校验控件的父容器是同一个。

除了 **hsButton** 按钮及其**派生控件**无须设置 TargetValidateContainer 属性就支持自动获取目标校验容器外，其他类型的按钮（如工具项按钮等）需要明确指定目标容器。

注意

HSRCF 目前支持的目标校验容器类型为 hsPanel。

2.) 编写按钮的自动校验后事件

在用户设置正确的属性，单击“保存”按钮时，框架会触发自动校验；但是如果需要在校验成功后做后续操作，如保存当前数据，就必须编写自动校验完成后的事件。

先绑定按钮的自动校验后事件，通过双击设计时属性窗中的事件名，编辑器将添加如下代码到后台文件：

```
        /// <summary>
        /// 按钮启用自动校验(EnableAutoValidate 属性为 True)后事件处理程序
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnOK_AfterAutoValidate(object sender,
AutoValidateResultEventArgs e)
        {


         }
```

然后在该方法中编写保存数据逻辑，伪代码如下所示：

```
      /// <summary>
      /// 按钮启用自动校验(EnableAutoValidate 属性为 True)后事件处理程序
```

```
     /// </summary>
     /// <param name="sender"></param>
     /// <param name="e"></param>
     private void btnOK_AfterAutoValidate(object sender,
AutoValidateResultEventArgs e)
    {
          //如果校验通过
          if (e.Result)
          {
              //保存数据
              //DoSaveData
              MessageUtility.ShowTips("保存成功！");
          }
     }
```

至此，整个校验完成。

12.6 注意事项

统一校验本身不针对恒生系列控件的，但即时校验（单一即时校验和循环即时校验）是针对恒生系列控件的；自动校验基于统一校验和即时校验，因此它是针对恒生系列控件的。

13 界面按键控制

如果编写的界面从插件基类(AddInPartBase)和窗体基类(AddInFormBase)继承而来，那么框架将控制键盘按键在界面的行为，其控制方式如下：

按键	描述	控制行为
UP	向上键	将按照 Tab 索引顺序跳转到上一个 Tab 索引顺序的控件。
Down	向下键	将按照 Tab 索引顺序跳转到下一个 Tab 索引顺序的控件。
Enter	回车键	将按照 Tab 索引顺序跳转到下一个 Tab 索引顺序的控件。

注意

- 恒生系列控件以及从恒生系列控件继承的控件默认支持按键跳转。
- 恒生系列需要控制跳转的控件提供 **IsSkipNextControl** 属性（即是否跳转到下个控件的属性），以便快速设置该控件是否可跳转。
- 如果不是恒生系列控件，默认不支持跳转。
- 此外如果控件实现 IHSSkipSupport 接口，就按照该接口控制，否则就通过以上三点控制。
- 如果控件不跳转，表示框架跳过该控件的按键控制，即可以自由处理控件的上下键、回车键等，按原来方法处理。

13.1 IHSSkipSupport 接口

只需在待自定义跳转行为的控件中实现该接口即可。

该接口声明如下所示：

```
/// <summary>
/// HS 按键跳转支持，目前支持上下键，回车键跳转
```

```
    /// </summary>
    public interface IHSSkipSupport
    {
        /// <summary>
        /// 是否通过上键跳转
        /// </summary>
        bool IsSkipByUp { get; }
        /// <summary>
        /// 是否通过下键跳转
        /// </summary>
        bool IsSkipByDown { get; }
        /// <summary>
        /// 是否通过回车键跳转
        /// </summary>
        bool IsSkipByEnter { get; }
    }
```

控件可以显式实现该接口，事件如下所示：

```
    //自定义按键跳转控制
    public class MySkipCustomControl : Control,  IHSSkipSupport
    {
        // 获取或设置一个值，该值指示是否跳转到上或下一个控件，默认是启用跳转。
        private bool isSkipNextControl = true;

        [Browsable(true)]
        [DefaultValue(true)]
        [Description("获取或设置一个值，该值指示是否跳转到上或下一个控件，默认是启用跳转。")]
        public bool IsSkipNextControl
        {
            get { return isSkipNextControl; }
            set { isSkipNextControl = value; }
        }

        // 是否控制回车键
        bool IHSSkipSupport.IsSkipByEnter
        {
            //根据这个属性进行控制
            get{ return this.IsSkipNextControl; }
        }

        // 是否通过向上键跳转
        bool IHSSkipSupport.IsSkipByUp
```

```
        {
            //根据这个属性进行控制
            get { return this.IsSkipNextControl; }
        }

        // 是否通过向下键跳转
        bool IHSSkipSupport.IsSkipByDown
        {
            //根据这个属性进行控制
            get { return this.IsSkipNextControl; }
        }

        //...更多代码
    }
```

14 多语言

HSRCF 的多语言支持主要分为框架的多语言开发和插件的多语言开发两个部分，这两个部分的多语言设计方案各不相同。

14.1 框架的多语言开发

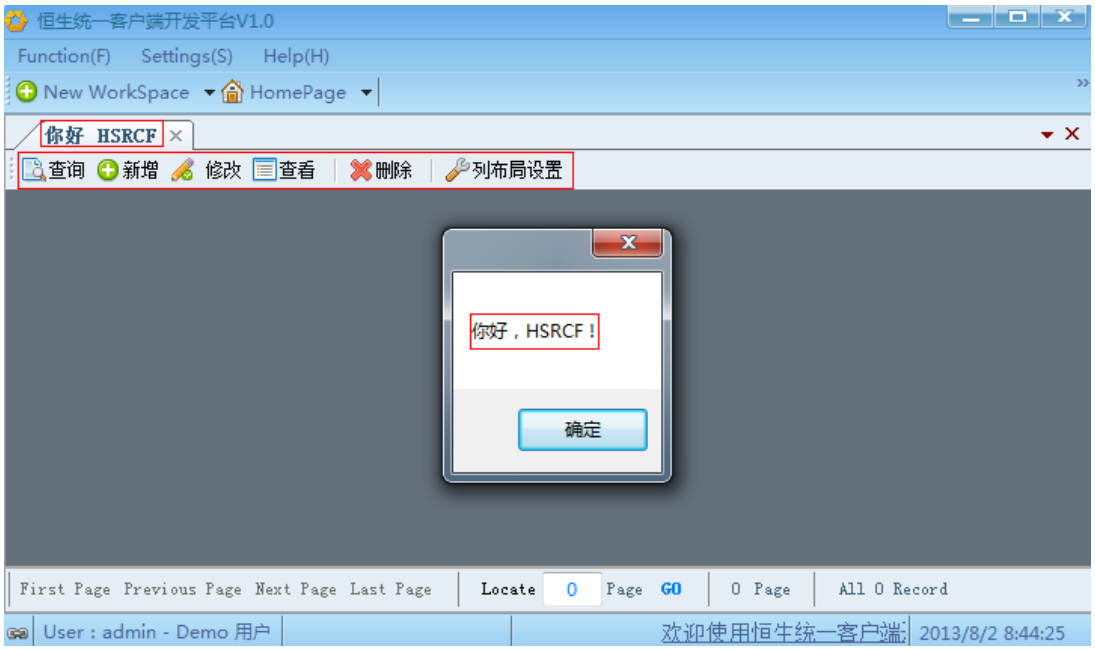
框架的多语言主要涉及到到菜单项、工具栏项、状态栏项的多语言设置。框架内容的多语言配置方法在开发和配置菜单栏一节中已经做了介绍。

14.2 插件的多语言开发

插件内容的多语言配置主要涉及到资源文件，使用.NET 框架提供的多语言开发支持，详细的多语言开发方法请参考.NET 多语言开发的文档。

下面以前面开发的 Hello HSRCF 插件的多语言支持开发为案例，介绍一下简单的多语言开发方法。

Hello HSRCF 插件主要需国际化的有三处：插件标签标题、界面组件名称（如工具按钮的名称）和 Hello HSRCF 字符串，如下图所示。

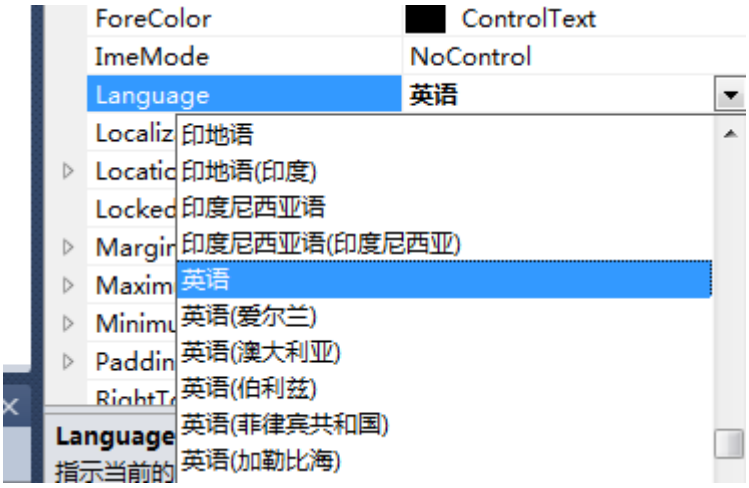


14.2.1 界面组件多语言

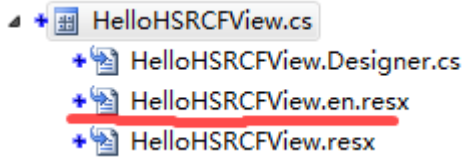
详细步骤如下：

1.) 打开 HelloHSRCFView.cs 控件，假设默认开发中文语言的系统，先将查询按钮的 Text 属性改为“查询”，这样显示的语言就是中文了。

2.) 在设计视图下，选中 HelloHSRCFView 插件，在属性窗口中，找到 Language 属性。选中英语选项：



3.) 此时选中查询按钮，将其 Text 属性改为“Query”， 点击保存后，可以在解决方案资源管理器窗口中看到一个 Resources.en.resx 文件，如下图所示。



该文件就是英文状态下对应的资源文件，双击打开可以看到刚才更改过的 Text 属性和其值，但是不要编辑该资源文件。

注意

设置了 Language 属性的设计视图中不能添加或删除控件，只有在默认的设计视图中才能添加或删除控件。返回默认视图的方法是将 Loalizable 属性设为 False。

14.2.2 消息多语言

详细步骤如下：

1.) 对于无法在视图中更改的多语言文字，比如硬编码在查询按钮的 Click 事件中的“你好，HSRCF！”字符串，需要用到自建的资源文件。在前面已经在 Properties 文件夹下面建立了 Resources.resx 文件，现在添加一个名为 Resources.en.resx 的文件。此文件就是对应的英文环境下的资源文件。在两个文件中分别加入对应的字符串：

Resources.resx 资源文件

名称	值
StrHello	你好，HSRCF！

Resources.en.resx 资源文件

名称	值
StrHello	Hello, HSRCF!

2.) 如何使用资源文件？微软已经将资源文件编译成相关的类，只要在 Click 中直接调用相关的类即可，如下所示：

```
MessageBox.Show(Properties.Resources.StrHello);
```

注意：Properties 类下是没有 Resources.en 这个类的，只要调用 Resources 类，程序会自动根据语言环境来调用相关的资源文件。

14.2.3 插件标题多语言

需要注意的是插件标签名称的多语言配置，在创建 HelloHSRCFView 插件时，已经在 Resources.resx 文件中创建了一个字符串资源。现在在 Resources.en.resx 中创建相应的英文资源。修改完以后，相应资源文件中的名称和值如下所示：

Resources.resx 资源文件

名称	值
HelloHSRCFView	你好 HSRCF

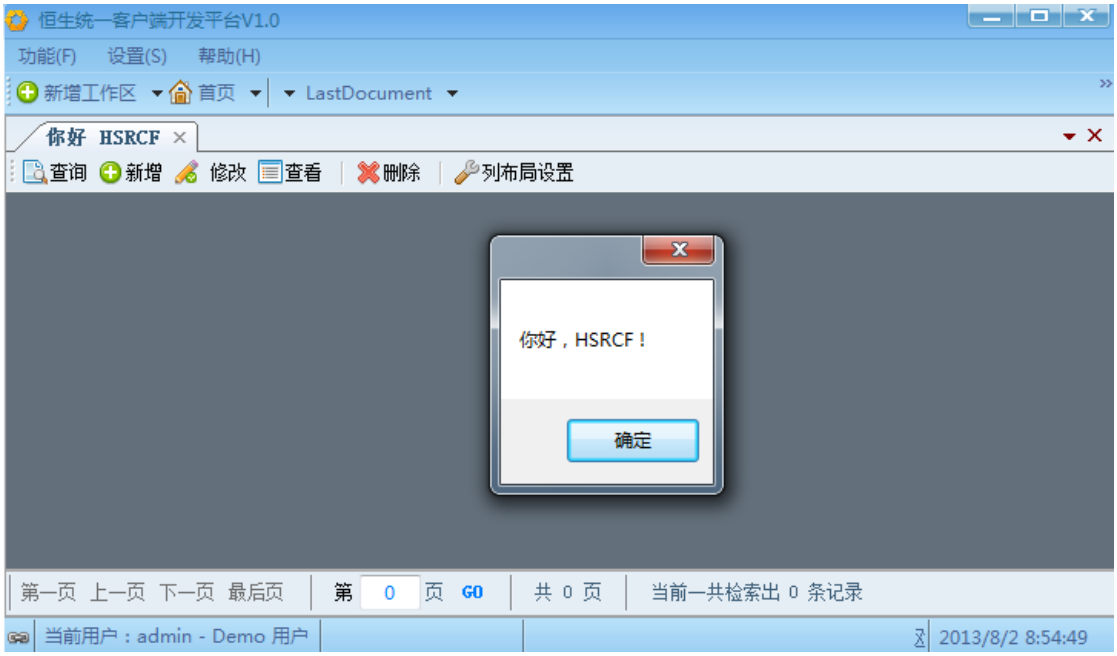
Resources.en.resx 资源文件

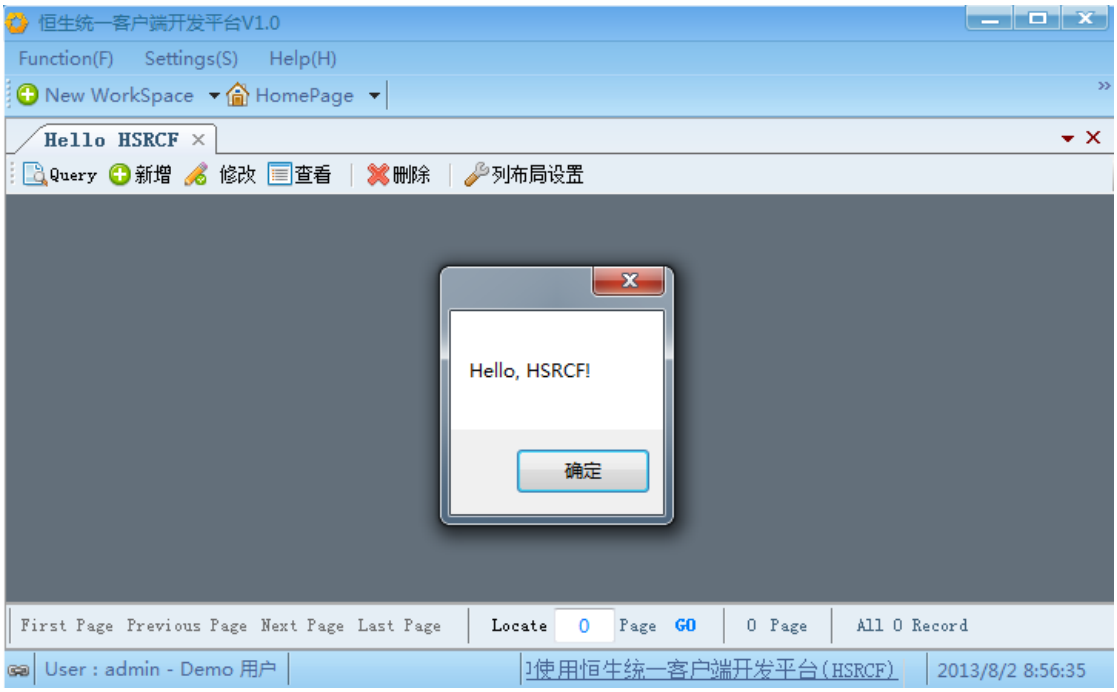
名称	值
HelloHSRCFView	Hello HSRCF

注意

资源文件中插件标签名的名称需要遵守一个约定：资源文件的命名必须是 Resources.resx。资源文件中的名称必须是插件类的名字，比如 HelloWorldView 插件的类名必须要在资源文件中，否则插件名称无法显示。

最后启动开发平台，打开 HelloHSRCFView 插件，切换语言环境(可以通过**设置->语言**)，就能看到相应效果，如下图所示。





说明

在英文环境下，对话框中显示的确定按钮名称为中文，是因为当前操作系统的语言环境是中文。如果操作系统的语言环境是英文，则显示英文。

14.3 Language 对照表

一些 Language 对照表如下，更详细的可以参考MSDN。

区域	Language	资源文件名称
英文	en	Resources.en.resx
英文(美国)	en-US	Resources.en-US.resx
英文(英国)	en-GB	Resources.en-GB.resx
中文(简体)	zh-CHS	Resources.zh-CHS.resx
中文(繁体)	zh-CHT	Resources.zh-CHT.resx
中文(中国)	zh-CN	Resources.zh-CN.resx
中文(香港)	zh-HK	Resources.zh-HK.resx

区域	Language	资源文件名称
中文(澳门)	zh-MO	Resources.zh-MO.resx
中文(台湾)	zh-TW	Resources.zh-TW.resx
中文(新加坡)	zh-SG	Resources.zh-SG.resx
日语(日本)	jr-JP	Resources.jr-JP.resx
朝鲜语(韩国)	ko-KR	Resources.ko-KR.resx
……		

15 常用接口和对象

15.1 常用工具类

常用工具类都存在于 Hundsun.Framework.Entity.dll 动态库中，可以引用该动态库，并且添加命名空间 Hundsun.Framework.Entity 进行使用。对于其内部的方法、属性，可以参考提供的 chm 类型的帮助文档。常用的工具类如下表所示。

工具类名	描述
BinaryUtility	二进制工具类
ConvertUtility	类型转换工具类
DataTableUtility	DataTable 工具类，提供如下功能： ● 同数据结构数据表的合并 ● 数组转 DataTable ● 枚举类型转换为 DataTable ● DataTable 转实体数组 ● 根据列名的列表创建一个表格结构
DirUtility	文件夹操作工具类
EnumUtility	枚举工具类
FileUtility	文件操作工具类
KeyboardUtility	键盘操作工具类，提供访问键盘当前状态的属性，和发送指定键的方法
MouseUtility	鼠标操作工具类，可以模拟鼠标单击（左右键）
PinYinUtility	中文拼音工具类
PrinterUtility	打印机相关工具类
SerializeUtility	序列化工具类

工具类名	描述
XYUtility	处理坐标转换的类

15.2 插件相关

接口/类	描述
IAddInContainer	主窗口实现的接口，通过它可以直接或间接访问框架内部所有公开的接口，需要添加 Hundsun.Framework.AddIn.dll 引用。
IExecuter	启动 WorkItem 的接口，通过它可以把一些全局的接口对象注入到依赖容器，需要添加 Hundsun.Framework.AddIn.dll 引用。
IAddContent	插件停靠容器的接口，通过它可以指定标识的插件可以放到指定的容器中，需要添加 Hundsun.Framework.AddIn.dll 引用。
AddInPartAttribute	插件特性，通过它可以把指定的用户控件标识为框架能够识别的插件，需要添加 Hundsun.Framework.AddIn.dll 引用
AddInParameterAttribute	插件参数特性，通过它可以把指定的插件内相关参数标识为框架能够识别的参数，需要添加 Hundsun.Framework.AddIn.dll 引用。
AddInToolStripItemAttribute	插件 ToolStrip 特性，通过它可以把指定的 ToolStripItem 标识为框架能够识别的 ToolStripItem，需要添加 Hundsun.Framework.AddIn.dll 引用。
LinkageEventPublicationAttribute	插件主联动事件特性，通过它可以指定插件内的主联动事件，需要添加 Hundsun.Framework.AddIn.dll 引用。
LinkageEventSublicationAttribute	插件被联动方法属性，通过它可以指定插件内的被联动方法，需要添加 Hundsun.Framework.AddIn.dll 引用。
ProressEventAttribute	插件进度属性，通过它可以指定插件内可以对事件应用属性，需要添加 Hundsun.Framework.AddIn.dll 引用。

15.3 依赖注入相关

接口/类	描述
DependencyAttribute	用于指定一个属性或参数存在一个依赖关系，并且在将所在类的实例添加到 WorkItem 时，必须解决依赖关系，需要添加 Hundsun.Framework.MVP.dll 引用。
OptionaDependencyAttribute	可选的依赖参数的特性的基类，需要添加 Hundsun.Framework.MVP.dll 引用。
Component DependencyAttribute	用于声明属性、构造器或方法的参数是对父 WorkItem 的组件集合（WorkItem.Items）的组件的引用，需要添加 Hundsun.Framework.MVP.dll 引用。
ServicesDependencyAttribute	用于指定一个属性或参数是一个对服务的依赖，并且在将所在类的实例添加到 WorkItem 时，必须解决依赖关系，需要添加 Hundsun.Framework.MVP.dll 引用。
StateAttribute	指定一个属性或参数必须从 WorkItem 的 State 的值注入，需要添加 Hundsun.Framework.MVP.dll 引用。
ServiceAttribute	指定一个类为根 WorkItem 的服务。在装载模块时会被自动注入到 WorkItem 中，需要添加 Hundsun.Framework.MVP.dll 引用。

15.4 通信和数据相关

数据通信相关接口和类都存在于 Hundsun.Framework.Communication.dll 动态库中，可以引用该动态库，并且添加命名空间 Hundsun.Framework.Communication 进行使用。对于其内部的方法和属性，可以参考提供的 chm 类型的帮助文档。下面根据功能介绍一下接口和类。

15.4.1 主要接口

接口	描述
ICommunication	数据通信接口，它用于获取数据工厂接口（IDBFactory）实例
IDBFactory	数据工厂接口

类/枚举	描述
Communication	数据通信接口实现类，目前默认实现在 Hundsun.Framework.Client 项目中，并在系统启动后默认注入一该实例
DBFactory	数据工厂

15.4.2 T2 通信接口/类

接口	描述
IT2DataHandler	T2 数据通讯接口
IT2ESBMessage	T2 报文操纵接口
IT2Packer	T2 打包器接口
IT2UnPacker	T2 解包器接口

类/枚举	描述
T2DataHandler	T2 数据通讯实现类
T2ESBMessage	T2 报文
T2Packer	T2 打包器
T2UnPacker	T2 解包器
T2Field	T2 业务包体字段类
T2FieldType	T2 报文字段类型
T2PackVersion	T2 打包器版本
T2Exception	T2 异常类的基类
T2CommunicationException	T2 通讯异常类
T2EsbMessageException	T2 报文异常类
T2PackerException	T2 打包异常类
T2UnPackerException	T2 解包异常类

15.4.3 消息订阅通信接口/类

接口	描述
IMC2Subscriber	消息中心订阅者接口
IMC2Adapter	MC 消息中心适配器接口

类/枚举	描述
MC2Subscriber	消息中心订阅者
MC2Adapter	消息中心适配器
MC2Exception	消息中心异常类
MC2AdapterErrorNo	消息中心适配器错误枚举

15.4.4 客户端日志接口/类

接口	描述
ILogUtility	日志接口

类/枚举	描述
HSLogLevel	日志级别，值越大级别越高，级别越高错误越严重
HSLogType	日志类型
LogUtility	日志记录类
HSLog	日志静态类

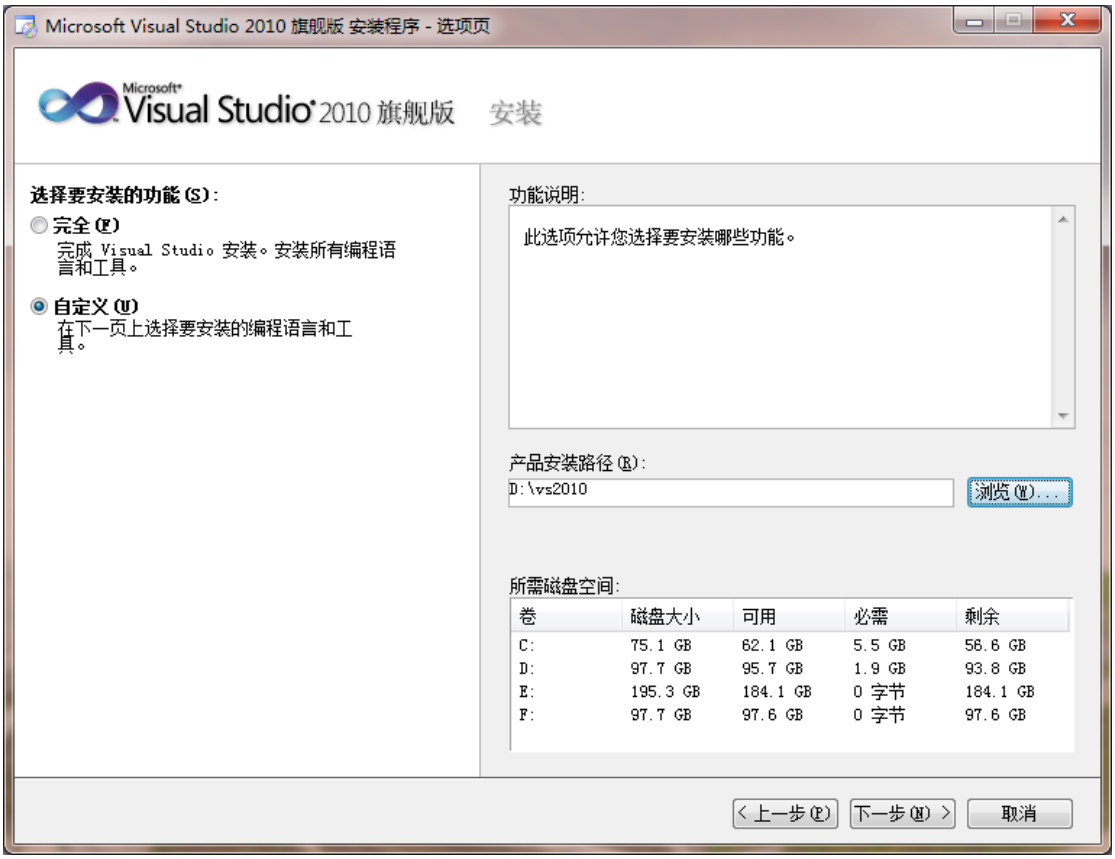
15.4.5 SQLLite 数据访问接口/类

接口/类	描述
ISQLiteDataHandler	SQLLite 数据访问接口
SQLiteDataHandler	SQLLite 数据访问实现类

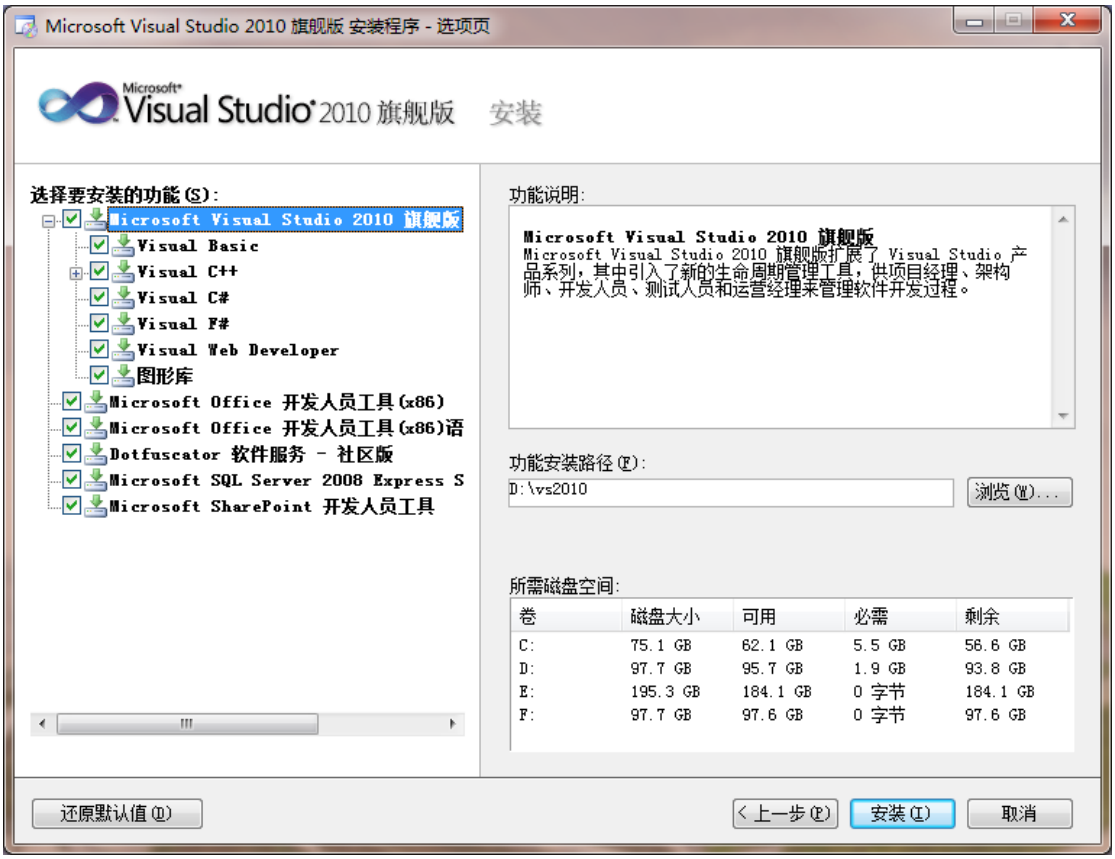
附录 A Visual Studio 安装

1.) 双击安装包中的 Setup.exe，弹出如下界面，然后直接单击第一个选项安装 Microsoft Visual Studio 2010：

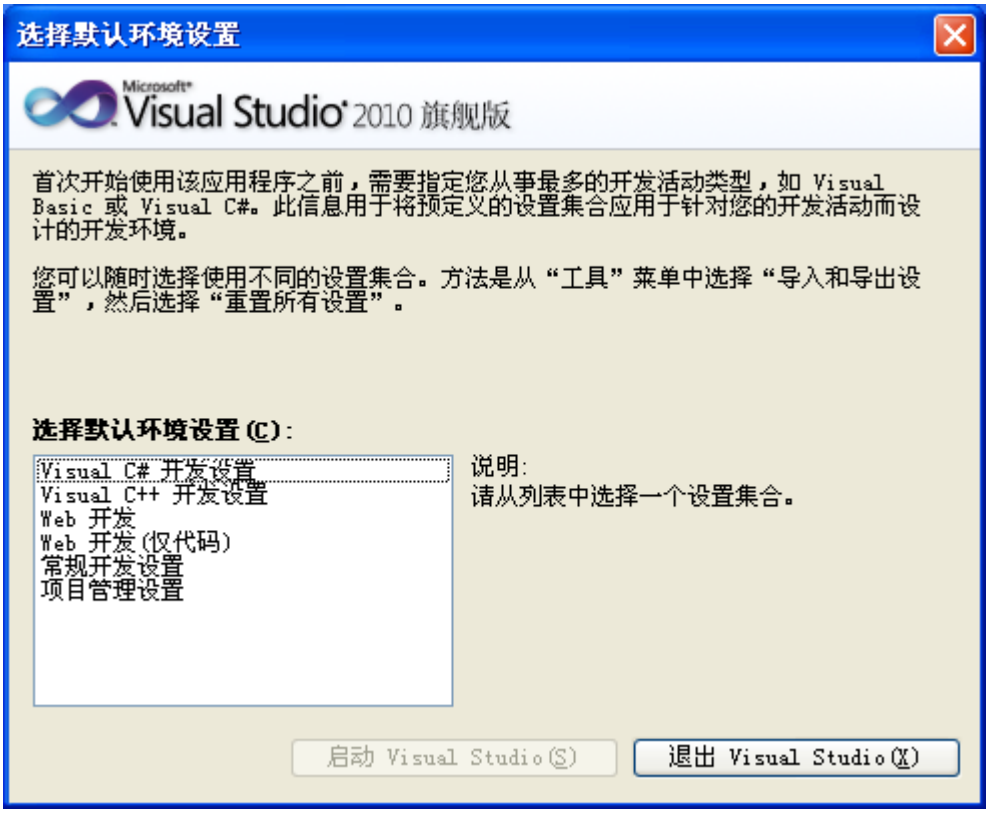
2.) 选择自定义安装，可以修改安装路径。


Microsoft Visual Studio 2010 旗舰版 安装程序 - 选项页
Visual Studio 2010 旗舰版 安装
选择要安装的功能(S):
完全(E)
完成 Visual Studio 安装。安装所有编程语言和工具。
自定义(U)
在下一页上选择要安装的编程语言和工具。
功能说明:
此选项允许您选择要安装哪些功能。
产品安装路径(R):
D:\vs2010
浏览(W)...
所需磁盘空间:
卷 磁盘大小 可用 必需 剩余
C: 75.1 GB 62.1 GB 5.5 GB 56.6 GB
D: 97.7 GB 95.7 GB 1.9 GB 93.8 GB
E: 195.3 GB 184.1 GB 0 字节 184.1 GB
F: 97.7 GB 97.6 GB 0 字节 97.6 GB
< 上一步(P)
下一步(N) >
取消

3.) 安装所有的功能模块：



4.) 安装成功后首次启动时，选择 C#开发设置。



5.) 显示源代码行号。

选择“工具(T)”→“选项(O)”→ “文本编辑器”，选择 C#，把显示里面的行号打好勾，这样编辑程序时，程序前面就会显示行号了，方便快速定位错误代码。

附录 B C# 语言和.NET Framework 介绍

C# 是一种简洁、类型安全的面向对象的语言，开发人员可以使用它来构建在 .NET Framework 上运行的各种安全可靠的应用程序。您可以使用 C# 来创建传统的 Windows 客户端应用程序、XML Web services、分布式组件、客户端/服务器应用程序、数据库应用程序等等。Visual Studio 提供了高级代码编辑器、方便的用户界面设计器、集成调试器和许多其他工具，使您可以更轻松地在 C# 语言 4.0 版和 .NET Framework 4 版的基础上开发应用程序。

C# 语言

C# 语法表现力强，而且简单易学。C# 的大括号语法使任何熟悉 C、C++ 或 Java 的人都可以立即上手。了解上述任何一种语言的开发人员通常在很短的时间内就可以开始使用 C# 高效地进行工作。C# 语法简化了 C++ 的诸多复杂性，并提供了很多强大的功能，例如可为 null 的值类型、枚举、委托、lambda 表达式和直接内存访问，这些都是 Java 所不具备的。C# 支持泛型方法和类型，从而提供了更出色的类型安全和性能。C# 还提供了迭代器，允许集合类的实施者定义自定义的迭代行为，以便容易被客户端代码使用。语言集成查询 (LINQ) 表达式使强类型查询成为了一流的语言构造。

作为一种面向对象的语言，C# 支持封装、继承和多态性的概念。所有的变量和方法，包括 Main 方法（应用程序的入口点），都封装在类定义中。类可能直接从一个父类继承，但它可以实现任意数量的接口。重写父类中的虚方法的各种方法要求 override 关键字作为一种避免意外重定义的方式。在 C# 中，结构类似于一个轻量类；它是一种堆栈分配的类型，可以实现接口，但不支持继承。

除了这些基本的面向对象的原理之外，C# 还通过几种创新的语言构造简化了软件组件的开发，这些结构包括：

- 封装的方法签名（称为“委托”），它实现了类型安全的事件通知。
- 属性，充当私有成员变量的访问器。
- 特性，提供关于运行时类型的声明性元数据。
- 内联 XML 文档注释。
- 语言集成查询 (LINQ)，提供了跨各种数据源的内置查询功能。

在 C# 中，如果必须与其他 Windows 软件（如 COM 对象或本机 Win32 DLL）交互，则可

以通过一个称为“互操作”的过程来实现。互操作使 C# 程序能够完成本机 C++ 应用程序可以完成的几乎任何任务。在直接内存访问必不可少的情况下，C# 甚至支持指针和“不安全”代码的概念。

C# 的生成过程比 C 和 C++ 简单，比 Java 更为灵活。没有单独的头文件，也不要求按照特定顺序声明方法和类型。C# 源文件可以定义任意数量的类、结构、接口和事件。

下列各项是其他 C# 资源：

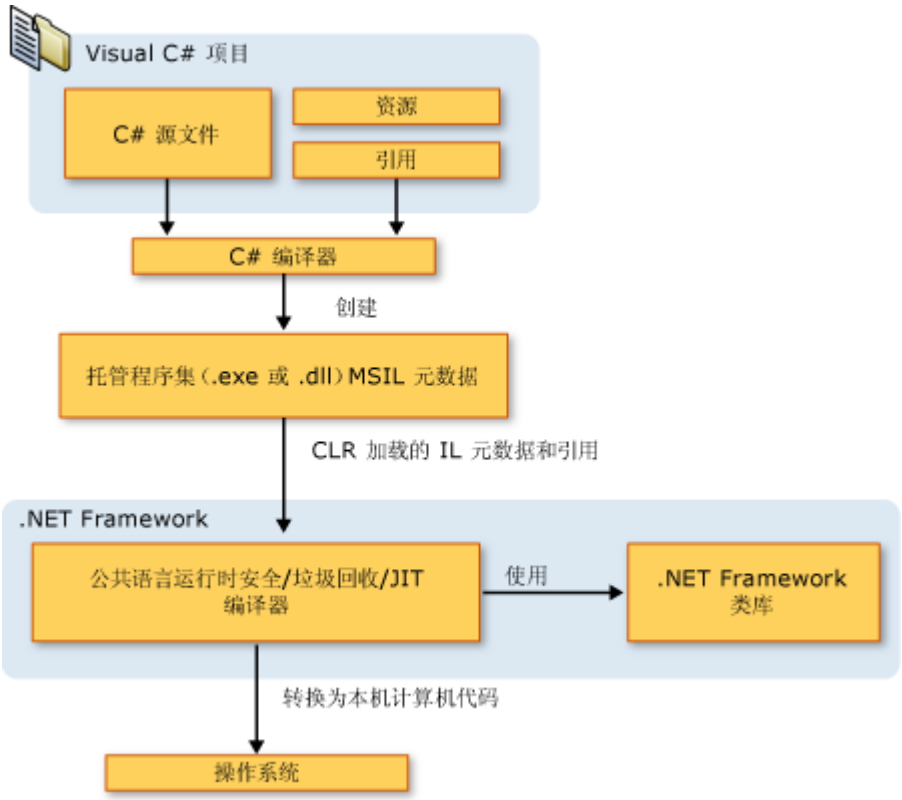
- 有关该语言的充分常规介绍，请参见 C# 语言规范 的第 1 章。
- 有关 C# 语言特定方面的详细信息，请参见 C# 参考。
- 有关 LINQ 的更多信息，请参见LINQ（语言集成查询）。
- 若要查找 Visual C# 团队提供的最新文章和资源，请访问 Visual C# 开发中心。

.NET Framework

C# 程序在 .NET Framework 上运行，它是 Windows 的一个不可或缺的组件，包括一个称为公共语言运行时 (CLR) 的虚拟执行系统和一组统一的类库。CLR 是 Microsoft 对 Common Language Infrastructure (CLI) 的商业实现。CLI 是一种国际标准，是用于创建语言和库在其中无缝协同工作的执行和开发环境的基础。

用 C# 编写的源代码被编译为一种符合 CLI 规范的中间语言 (IL)。IL 代码与资源（例如位图和字符串）一起作为一种称为程序集的可执行文件存储在磁盘上，通常具有的扩展名为 .exe 或 .dll。程序集包含清单，它提供有关程序集的类型、版本、区域性和安全要求等信息。

执行 C# 程序时，程序集将加载到 CLR 中，这可能会根据清单中的信息执行不同的操作。然后，如果符合安全要求，CLR 就会执行实时 (JIT) 编译以将 IL 代码转换为本机机器指令。CLR 还提供与自动垃圾回收、异常处理和资源管理有关的其他服务。由 CLR 执行的代码有时称为“托管代码”，它与编译为面向特定系统的本机机器语言的“非托管代码”相对应。下图阐释了 C# 源代码文件、.NET Framework 类库、程序集和 CLR 的编译时与运行时的关系。



语言互操作性是 .NET Framework 的一项主要功能。由于 C# 编译器生成的 IL 代码符合公共类型规范 (CTS)，因此从 C# 生成的 IL 代码可以与从 Visual Basic、Visual C++ 的 .NET 版本或者其他 20 多种符合 CTS 的语言中的任何一种生成的代码进行交互。单一程序集可能包含用不同 .NET 语言编写的多个模块，并且类型可以相互引用，就像它们是用同一种语言编写的。

除了运行时服务之外，.NET Framework 还包含一个由 4000 多个类组成的内容详尽的库，这些类被组织为命名空间，为从文件输入和输出、字符串操作、XML 分析到 Windows 窗体控件的所有内容提供了各种有用的功能。典型的 C# 应用程序使用 .NET Framework 类库广泛地处理常见的“日常”任务。

有关 .NET Framework 的更多信息，请参见 .NET Framework 概述。

附录 C Windows Form 介绍

Windows Form 是 Microsoft 提供开发 Windows 交付界面的应用程式开发平台，以 .NET Framework 为基础。这种架构提供清晰、面向对象的类库，让您能够开发各种 Windows 应用程式。

.NET Framework 是一组可简化常见应用程序任务的托管库，而 Windows 窗体则是其中的一种智能客户端技术。使用类似 Visual Studio 的开发环境时，您可以创建 Windows 窗体智能客户端应用程序，以显示信息、请求用户输入以及通过网络与远程计算机通信。

在 Windows Form 中，“窗体”是向用户显示信息的可视图面。通常情况下，通过向窗体上添加控件并开发对用户操作（如鼠标单击或按下按键）的响应，生成 Windows For 应用程序。“控件”是一个独立的用户界面 (UI) 元素，用于显示数据或接受数据输入。

当用户对窗体或其中的某个控件进行操作时，将生成事件。应用程序使用代码对这些事件进行响应，并在事件发生时处理事件。有关更多信息，请参见在 Windows 窗体中创建事件处理程序。

Windows Form 包含可添加到窗体上的各式控件：用于显示文本框、按钮、下拉框、单选按钮甚至网页的控件。有关您可以在窗体中使用的所有控件的列表，请参见在 Windows 窗体上使用的控件。Windows 窗体还支持使用UserControl类创建您自己的自定义控件，供您在某个现有控件不符合您的需要的情况下使用。

Windows Form 具有丰富的界面控件，可模拟类似 Microsoft Office 这样的高端应用程序中的功能。使用ToolStrip和MenuStrip控件时，可以创建包含文本和图像、显示子菜单及承载其他控件（如文本框和组合框）的工具栏和菜单。

使用 Visual Studio 的具有拖放功能的 Windows 窗体设计器，可以轻松创建 Windows Form 应用程序。只需使用光标选择控件并将控件添加到窗体上所需的位置即可。设计器提供类似网格线和对齐线的工具，可简化对齐控件的操作。无论使用 Visual Studio 还是在命令行上编译，都可以使用FlowLayoutPanel、TableLayoutPanel和SplitContainer控件以较短的时间创建高级窗体布局。

最后，如果您必须创建自己的自定义用户界面元素，则可使用System.Drawing命名空间，其中包含了大量的类，可供您选择用于直接在窗体上呈现线条、圆和其他形状。

更多请看 MSDN 上Windows Form 专题。