chrome Ui 框架学习笔记

邱金武

09 年 11 月 11 日

目录

学习 Chrome 源码也有一小段时间了，对该浏览器的 UI 部分小有了解，于是将自己的一些心得贴上来，希望对有需要的朋友有所帮助。

说明：

1. 本人所使用的 chrome 源码比较早，所以下载最新的代码多少会和文中有些差异。
2. 代码在 windows 下使用 visual studio 2008 编译。
3. 源码下载地址 (http://code.google.com/chromium/)

推荐阅读：
http://www.cnblogs.com/duguguiyu/archive/2008/10/02/1303095.html

Ui 部分的通用[1]代码主要在目录树的“src/chrome/views”目录下。其中

1. controls 目录封装用到的控件，例如 Label、Textfield 等
2. widget 目录主要封装系统相关的 UI 底层细节，特别是 UI 消息机制。
3. window 目录主要封装 UI Frame 相关的细节，例如窗口的标题栏、系统按钮以及 Frame、Dialog 等的代理[2]接口。

如果想了解 Chrome 绘图技术，可以去 Skia 项目[3]看看

1. 一个简单的程序

```
//test.h
#include "chrome/views/view.h"
#include "chrome/views/window/window_delegate.h"

class TestWindow :
        public views::View,
        public views::WindowDelegate{
public:

       virtual View* GetContentsView() {
             return this;
       }

        virtual gfx::Size GetPreferredSize(){
              return gfx::Size(400,300);
        }

        static void CreateTestWindow();

};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//test.cpp
#include "test.h"
#include "chrome/views/window/window.h"

void TestWindow::CreateTestWindow(){
        views::Window::CreateChromeWindow(NULL,gfx::Rect(),new TestWindow)->Show();
}
```

1 说“通用”是因为 chrome 的主窗口框架是单独定制的。
2 设计模式术语
3 http://code.google.com/p/skia/

运行的结果如下：[4]

1.1. 源码分析

1. TestWindow 需要继承自两个类 views::View 和views::WindowDelegate。
   1. 在chrome中，任何控件[5]包括一个Frame都必须是一个View。
   2. 一个顶层窗口（例如Frame、Dialog[6]）必须继承自views::WindowDelegate以便传递该窗口的一些参数,例如窗口的尺寸。
2. GetPreferredSize()含义很明显是告诉系统初始窗口大小[7]为（400*300），这里其实可选。
3. GetContentsView()函数将自己作为一个窗口的根View传给所附属的Frame（或Dialog）。
4. 中间的黑色是因为客户区没有任何东西，所以背景贴图和背景贴图未覆盖的地方被显示出来。
5. 内部的白色框是因为系统默认会在客户区画白色的边框。

为了方面其他代码调用，这里提供一个静态函数CreateTestWindow()打开该窗口

2. UI 消息机制[8]

Windows 每一个控件都单独处理消息，chrome一个整的的窗口可以看做一个[9]（Windows）控件，所以所有控件的消息都会发到一块去，所以chrome Ui框架有一套机制来

4 因为本人已替换 chrome 自带的 frame 贴图，所以风格和 chrome 并不一致。
5 这里忽略操作系统原生控件的某些特殊情况
6 Dialog 继承自 DialogDelegate，该代理继承自 WindowDelegate
7 确切的说是窗口内部的根 View 的大小，实际窗口大小要大于该值。
8 针对 windows 平台
9 这里忽略原生控件的封装

分发到具体的（Chrome）控件。

Windows版本的chrome的UI部分基于WTL[10]。chrome通过在此处[11]中的绑定来获取windows UI消息。接着Chrome内置的一套消息分发机制将该消息发送到具体的chrome 控件。

3. Chrome 控件树

前面提到，chrome每一个控件都是一个views::View。views::View可以认为是chrome中所有控件的基类。在这个类中定义了一个通用的操作和默认实现，例如鼠标单击处理函数，如果某控件需要自定义鼠标处理事件，可以在自己的类中覆盖基类的默认实现。

每一个views::View可以包含子views::View。所以每一个都包含一个父views::View和若干子views::View的指针，下面是源码[12]的定义

```
// This view's parent
View *parent_;

// This view's children.
typedef std::vector<View*> ViewList;
ViewList child_views_;
```

这棵树的根节点便是views::RootView[13]。这个View的主要用途就是从views::Widget[14]中接收UI消息。

这棵树的最初几层可以参考chrome源码[15]

```
////////////////////////////////////////////////////////////////////////////////
// NonClientView
//
//  The NonClientView is the logical root of all Views contained within a
//  Window, except for the RootView which is its parent and of which it is the
//  sole child. The NonClientView has two children, the NonClientFrameView which
//  is responsible for painting and responding to events from the non-client
//  portions of the window, and the ClientView, which is responsible for the
//  same for the client area of the window:
//
//  +- views::Window ------------------------------------+
//  | +- views::RootView ----------------------------+ |
//  | | +- views::NonClientView -------------------+ | |
//  | | | +- views::NonClientView subclass     ---+ | | |
//  | | | |                                       | | | |
//  | | | | << all painting and event receiving >> | | | |
//  | | | | << of the non-client areas of a     >> | | | |
//  | | | | << views::Window.                   >> | | | |
//  | | | |                                       | | | |
//  | | | +---------------------------------------+ | | |
//  | | | +- views::ClientView or subclass --------+ | | |
//  | | | |                                       | | | |
//  | | | | << all painting and event receiving >> | | | |
//  | | | | << of the client areas of a         >> | | | |
//  | | | | << views::Window.                   >> | | | |
//  | | | |                                       | | | |
//  | | | +---------------------------------------+ | | |
//  | | +-------------------------------------------+ | |
//  | +-----------------------------------------------+ |
```

10 http://sourceforge.net/projects/wtl/
11 src\chrome\views\widget\widget_win.h
12 src\chrome\views\view.h
13 src\chrome\views\widget\root_view.h
14 src\chrome\views\widget\widget.h
15 src\chrome\views\window\non_client_view.h

```
// | +-------------------------------------------------+ |
// +-----------------------------------------------------+
//
// The NonClientFrameView and ClientView are siblings because due to theme
// changes the NonClientFrameView may be replaced with different
// implementations (e.g. during the switch from DWM/Aero-Glass to Vista Basic/
// Classic rendering).
```

说明：

1. views::Window 表示一个实际的窗口，不过它并不是一个views::View。
2. views::RootView 如前面所述，即整个控件树的根。
3. Views::NonClientView 表示整个实际的控件树的根，实际上 views::RootView 只有一颗子树 Views::NonClientView，所以个人认为这两个 View 可以合并在一块。也许是为了从逻辑上分开吧，即 RootView 向上和 OS 层打交道，而 NonClientView 则专注于下层的控件 View
4. views::NonClientView subclass 表示整个非客户区的 View。例如窗口的标题栏、窗口图标、关闭按钮、最大化/最小化按钮、边框等就定义在此。实际上 chrome 默认使用 views::NonClientView 的子类 CustomFrameView [16]。此外 chrome 还提供原生 View 的封装 NativeFrameView[17]。
5. views::ClientView or subclass 是窗口客户区的根，上面 test 实例中的 TestWindow 也是一个 views::View。该 View 就是挂在【views::ClientView or subclass】的下面[18]。具体的挂在通过函数 GetContentsView() 实现。
6. chrome 通过创建 Popup[19]窗口将原生的标题栏、边框等组件隐藏起来，然后自己来原生的客户端自己来弄一套“非客户区”。

4. Chrome 消息分发机制

下面以一个鼠标移入事件说明，views::View 中定义了函数 OnMouseEntered，当鼠标移入某控件时，该函数被调用。下面是相关的函数声明。

```
// This method is invoked when the mouse enters this control.
//
// Default implementation does nothing. Override as needed.
virtual void OnMouseEntered(const MouseEvent& event);
```

当views::RootView收到windows消息时，例如鼠标移动的消息，它便会根据鼠标的位置获取具体的View，然后调用该View的相关处理函数。

因为相关的处理函数都在views::View中有一份默认实现，所以不关系该事件的View可以不理会。而如果需要定制处理函数只需重载相关的处理函数即可。

Chrome 查找具体的 View 通过 GetViewForPoint 函数，递归调用。View 首先查找所有子 View，如果事件的位置在某个子 View 的区域内，则调用该子 View 的 GetViewForPoint 函数。否则返回自己。下面是 RootView 分发“鼠标进入”事件的源码

16 src\chrome\views\window\custom_frame_view.h
17 src\chrome\views\window\native_frame_view.h ，该类本人未作测试。
18 作为它的子树
19 Windows 下可以参考 http://msdn.microsoft.com/en-us/library/ms632680%28VS.85%29.aspx

```
void RootView::OnMouseMoved(const MouseEvent& e) {
   View* v = GetViewForPoint(e.location());
   // Find the first enabled view.
   while (v && !v->IsEnabled())
      v = v->GetParent();
   if (v && v != this) {
      if (v != mouse_move_handler_) {
         if (mouse_move_handler_ != NULL) {
            MouseEvent exited_event(Event::ET_MOUSE_EXITED, 0, 0, 0);
            mouse_move_handler_->OnMouseExited(exited_event);
         }

         mouse_move_handler_ = v;

         MouseEvent entered_event(Event::ET_MOUSE_ENTERED,
                                  this,
                                  mouse_move_handler_,
                                  e.location(),
                                  0);
         mouse_move_handler_->OnMouseEntered(entered_event);
      }
```

5. 基本控件

前面提到过，chrome 的基础控件在目录“ src/chrome/views/controls”目录下。下面是 chrome 自带的控件一览

```
//按钮
D:\chrometrunk\chrometrunk\src\chrome\views\controls\button>ls
button.cc          custom_button.cc  native_button.cc       radio_button.h
button.h           custom_button.h   native_button.h        text_button.cc
button_dropdown.cc  image_button.cc   native_button_win.cc    text_button.h
button_dropdown.h   image_button.h    native_button_win.h
checkbox.cc        menu_button.cc    native_button_wrapper.h
checkbox.h         menu_button.h     radio_button.cc

//菜单
D:\chrometrunk\chrometrunk\src\chrome\views\controls\menu>ls
chrome_menu.cc  controller.h  menu.h
chrome_menu.h   menu.cc       view_menu_delegate.h

//滑动条
D:\chrometrunk\chrometrunk\src\chrome\views\controls\scrollbar>ls
bitmap_scroll_bar.cc  native_scroll_bar.cc  scroll_bar.cc
bitmap_scroll_bar.h   native_scroll_bar.h   scroll_bar.h

//表格
D:\chrometrunk\chrometrunk\src\chrome\views\controls\table>ls
group_table_view.cc  table_view.cc  table_view_unittest.cc
group_table_view.h   table_view.h

//树
D:\chrometrunk\chrometrunk\src\chrome\views\controls\tree>ls
tree_model.h         tree_node_iterator_unittest.cc  tree_view.cc
tree_node_iterator.h  tree_node_model.h              tree_view.h

//其他控件
```

```
D:\chrometrunk\chrometrunk\src\chrome\views\controls>ls
button        label_unittest.cc     native_control_win.h  tabbed_pane.h
combo_box.cc  link.cc               scroll_view.cc        table
combo_box.h   link.h                scroll_view.h         text_field.cc
hwnd_view.cc  menu                  scrollbar             text_field.h
hwnd_view.h   message_box_view.cc   separator.cc          throbber.cc
image_view.cc message_box_view.h    separator.h           throbber.h
image_view.h  native_control.cc     single_split_view.cc  tree
label.cc      native_control.h      single_split_view.h
label.h       native_control_win.cc tabbed_pane.cc
```

尽管 chrome 提供丰富的控件，但是如果打算使用 chrome 这一套 UI 框架开发自己的程序，这些远远不够用，幸运的是基于 chrome 开发自定义控件相当的方便。

接着上面的例子，下面的程序在客户区添加一个 Button 和 Label。当点击按钮后，Label 显示“点击了“。下面是源码：

```
//test.h
#include "chrome/views/view.h"
#include "chrome/views/window/window_delegate.h"
#include "chrome/views/controls/button/button.h"

namespace views{
        class Label;
        class TextButton;
}
class ChromeCanvas;


class TestWindow :
        public views::View,
        public views::WindowDelegate,
        public views::ButtonListener{
public:

        TestWindow();

       virtual View* GetContentsView() {
             return this;
       }

        virtual gfx::Size GetPreferredSize(){
              return gfx::Size(400,300);
        }

        virtual void Layout();
        virtual void Paint(ChromeCanvas* canvas);
        virtual void ButtonPressed(views::Button* sender);

        static void CreateTestWindow();

private:

        views::Label * lable_;
        views::TextButton * button_;

};

//test.cpp
#include "test.h"
#include "chrome/views/window/window.h"
#include "chrome/views/controls/button/text_button.h"
#include "chrome/views/controls/label.h"
#include "chrome/common/gfx/chrome_canvas.h"
```

```
TestWindow::TestWindow(){
	this->lable_ = new views::Label(L"");
	this->button_ = new views::TextButton(this,L"点击");
	AddChildView(this->lable_);
	AddChildView(this->button_);
}

void TestWindow::Layout(){
	this->lable_->SetBounds(10,10,50,30);
	this->button_->SetBounds(10,50,60,30);
}


void TestWindow::Paint(ChromeCanvas* canvas){
	canvas->FillRectInt(SK_ColorWHITE, 0, 0, width(), height());
}

void TestWindow::ButtonPressed(views::Button* sender){
	if(sender == this->button_){
		this->lable_->SetText(L"点击了");
	}
}

void TestWindow::CreateTestWindow(){
	views::Window::CreateChromeWindow(NULL,gfx::Rect(),new TestWindow)->Show();
}
```

结果如下：

5.1. 源码解析

1. 包含相关的控件 views::Label * lable_ ， views::TextButton * button_;
2. 通过实现 views::ButtonListener 接口来处理 button_的点击事件
3. 使用 AddChildView 将两个控件添加为子控件。
4. 在 buttonPressed 函数中添加事件处理的代码。
5. Paint 函数将整个客户区画成白色
6. Layout 函数设置子控件的位置。

5.2. Paint&&Layout

Chrome 使用 Skia 作为二位绘图引擎，而绘图体现在每一个 View 的 Paint 函数中，views::View 的 Paint 函数屁事都不干，所以在第一个程序中可以看到客户区将背景图片和默认的黑色显示出来，而第二个函数，我们覆盖默认的 Paint 函数并且将整个 View（TestWindow）的区间绘成白色。

每一个 View 的 Layout 函数用于定位它所有的子 View。当因为某些原因整个窗口需要重构时，根 View 调用 Layout 函数，然后递归调用每一个子 View 的 Layout 函数。关于 Chrome 的 Layout，chrome 还提供一个 GridLayout[20]和 FillLayout[21]。具体的用法可以参考 chrome 中的源码。

5.3. 事件处理

chrome 控件通常的事件处理方式是通过声明一个接口，然后将该接口指针作为一个成员变量来使用实现。例如按钮事件的接口声明如下：

```
// An interface implemented by an object to let it know that a button was
// pressed.
class ButtonListener {
  public:
   virtual void ButtonPressed(Button* sender) = 0;
};
```

一般情况下，控件的父 View 会继承子控件依赖的借口并实现相关的函数，然后将自己作为参数传递给这些控件实现完整的事件处理。一个父 View 可能包含好几个同类的控件，所以这些接口一般会包含一个 sender 以便区分不同的控件发送者。

其他控件的使用，可以参考源码中的代码，这里不多叙述。

6. 原生控件

一般来说，如果有设计良好，使用方便的控件可用就完全不必自己从轮子造起，chrome 就是这样。chrome 提供的很多控件并不是自己从零绘制的，而是基于原生的操作系统控件，最典型的是 TextField[22]。该控件底层的实现完全使用 WTL 提供的 richedit[23]。当然为了方便扩展，chrome 也提供相关的类方便用户封装其他的原生控件。

下面以封装 IE 浏览器说明问题，当用户需要在 UI 中嵌入浏览器，可以将下面的 View 像上面的代码中 Button 那样使用。

首先获得 WTL 对 IE 的封装（实际上是一个 com），相关源码可参考下面的代码（http://devel.openocr.org/svn/openocr/trunk/cuneiform/interface/icrashreport/wtl/samples/tabbrowser/browserview.h ）。

接着新建一个 View，并封装该原生控件，具体的源码可参考

```
// ie_view.h    BrowserView.h 是原生控件的封装头文件
#include "BrowserView.h"
#include "string"
#include "chrome/views/view.h"
```

20 src\chrome\views\grid_layout.h
21 src\chrome\views\fill_layout.h
22 src\chrome\views\controls\text_field.h
23 http://msdn.microsoft.com/en-us/library/bb787873%28VS.85%29.aspx#_win32_Rich_Edit_Version_2.0

```
#include "base/scoped_ptr.h"

namespace views{
	class HWNDView;
}

class IEView :
	public views::View{

public:

	explicit IEView(const std::wstring & url);
	virtual ~IEView();

	virtual void Layout();
	virtual void ViewHierarchyChanged(bool is_add,
		views::View* parent,
		views::View* child);

private:

	void initChildViews();

	scoped_ptr<CBrowserView> iebrowser_;
	views::HWNDView	* iebrowser_view_;

	std::wstring url_;

protected:

	DISALLOW_COPY_AND_ASSIGN(IEView);
};

// ie_view.cpp
#include "ie_view.h"
#include "base/logging.h"
#include "chrome/views/controls/hwnd_view.h"
#include "chrome/views/widget/widget.h"

IEView::IEView(const std::wstring & url) :
		url_(url) {
}

IEView::~IEView() {
}

void IEView::Layout() {
	this->iebrowser_view_->SetBounds(0,0,width(),height());
}



void IEView::initChildViews() {

	iebrowser_.reset( new CBrowserView );
	HWND hwnd = GetWidget()->GetNativeView();

	RECT rc = {0,0,100,100};
	HWND hwnd_ie = iebrowser_->Create( hwnd, 0, url_.c_str(), WS_CHILD|WS_VISIBLE | WS_CLIPSIBLINGS |
WS_CLIPCHILDREN | WS_VSCROLL | WS_HSCROLL);
	DWORD errcode = GetLastError();
	DCHECK(iebrowser_->IsWindow());

	iebrowser_view_ = new views::HWNDView;
	DCHECK(iebrowser_view_) << "LocationBarView::Init - OOM!";
	AddChildView(iebrowser_view_);
```

```
        iebrowser_view_->SetAssociatedFocusView(this);
        iebrowser_view_->Attach(iebrowser_->m_hWnd);

}


void IEView::ViewHierarchyChanged(bool is_add,
                                  views::View* parent,
                                  views::View* child) {
      if (is_add && child == this ) {
            this->initChildViews();
      }
}
```

这个过程中，最关键的 View 就是 views::HWNDView。该 View 提供一个机制将原生控件和 Chrome View 进行绑定，以便用户能够像 View 一样操作原生控件。

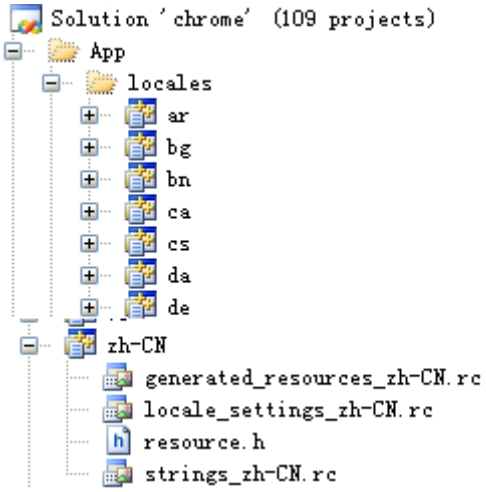
ViewHierarchyChanged函数在它所述的View被add和remove时被调用。上面源码的意思是当 IEView被父View调用AddChildView函数将其加为子View时被调用。之所以不放在构造函数内是因为有些控件的初始化依赖一个窗口句柄，而在构造函数结束之前，这个句柄很可能还没有初始化。

7. 国际化

不可否认，Chrome 的国际化做的非常优秀，在 Chrome 中添加一种新的语言支持非常方便。

7.1. Locale 项目

如果使用 virtual studio 2008 打开 chrome for windows 的工程，可以看到如下的项目：



其中每一个项目对应一种语言支持，所以如果需要添加新的语言支持，只需要新建一个新的语言项目。

实际上，每一个语言项目内的所有文件都是编译生成的中间文件，在文件夹【src\chrome\app\locales】中存放了这所有的项目文件，但每一个项目仅仅存在一个 vcproj

文件，例如 zh-CN.vcproj，项目中包含的文件实际存在于【src\chrome\Debug/Release\grit_derived_sources】目录下。那这些文件怎么生成的呢，这需要了解一下 google 自己开发的一个 python 项目。

7.2. GRIT 软件

该项目的源码在目录【src\tools\grit】下，全部使用 python 语言。该目录下的 readme 是这么说的：

```
GRIT (Google Resource and Internationalization Tool) is a tool for Windows
projects to manage resources and simplify the localization workflow.
```

在命令行输入命令【python grit.py[24]】有如下输出

```
GRIT - the Google Resource and Internationalization Tool
Copyright (c) Google Inc. 2009

Usage: grit [GLOBALOPTIONS] TOOL [args to tool]

Global options:

  -i INPUT  Specifies the INPUT file to use (a .grd file).  If this is not
            specified, GRIT will look for the environment variable GRIT_INPUT.
            If it is not present either, GRIT will try to find an input file
            named 'resource.grd' in the current working directory.

  -v        Print more verbose runtime information.

  -x        Print extremely verbose runtime information.  Implies -v

  -p FNAME  Specifies that GRIT should profile its execution and output the
            results to the file FNAME.

Tools:

  TOOL can be one of the following:
    build       A tool that builds RC files for compilation.
    newgrd      Create a new empty .grd file.
    rc2grd      A tool for converting .rc source files to .grd files.
    transl2tc   Import existing translations in RC format into the TC
    sdiff       View differences without regard for translateable portions.
    resize      Generate a file where you can resize a given dialog.
    unit        Use this tool to run all the unit tests for GRIT.
    count       Exports all translateable messages into an XMB file.

  For more information on how to use a particular tool, and the specific
  arguments you can send to that tool, execute 'grit help TOOL'
```

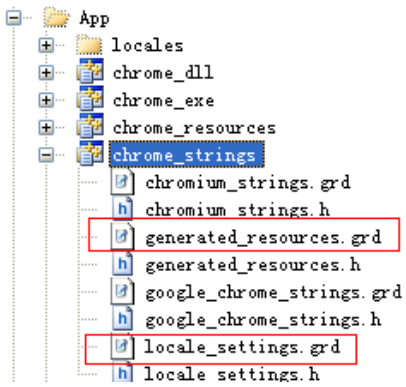
有兴趣可以研究一下代码，python 还是很有趣的东东！！！

grit 接收一个输入文件，然后生成项目所需的.h 和.rc 等文件。当然输出什么文件需要用户在输入文件中指定。

24 该文件是整个 grit 的入口

7.3. Grd 文件

grit 接收 grd 类型的文件作为输入，然后根据输入文件中的指定输出匹配的文件。locale 相关的输入文件存放在目录【src\chrome\app\resources】下，最关键的几个文件是【locale_settings.grd】和【generated_resources.grd[25]】。在 chrome 中有若干项目仅仅包含 grd 文件并将其生成目标文件，而其他一些项目则依赖这些文件。chrome_strings 项目就如此。它就将相关的 grd 文件生成 locale 下各种语言项目依赖的.h 和.rc 文件。



所有 grd 文件都是一个 xml 文件，格式都符合 grit 的一个规范，下面是【generated_resources.grd】的部分内容：

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- This file contains definitions of resources that will be translated for
each locale. -->

<grit base_dir="." latest_public_release="0" current_release="1"
        source_lang_id="en" enc_check="möl">
  <outputs>
    <output filename="grit/generated_resources.h" type="rc_header">
      <emit emit_type='prepend'></emit>
    </output>
    <output filename="generated_resources_zh-CN.rc" type="rc_all" lang="zh-CN" />
    <output filename="generated_resources_zh-TW.rc" type="rc_all" lang="zh-TW" />
    <output filename="generated_resources_zh-CN.pak" type="data_package" lang="zh-CN" />
    <output filename="generated_resources_zh-TW.pak" type="data_package" lang="zh-TW" />
  </outputs>
  <translations>
    <file path="resources/generated_resources_zh-CN.xtb" lang="zh-CN" />
    <file path="resources/generated_resources_zh-TW.xtb" lang="zh-TW" />
  </translations>
  <release seq="1" allow_pseudo="false">
    <messages fallback_to_english="true">
      <!-- TODO add all of your "string table" messages here.  Remember to
      change nontranslateable parts of the messages into placeholders (using the
      <ph> element).  You can also use the 'grit add' tool to help you identify
      nontranslateable parts and create placeholders for them. -->

      <message name="IDS_SHOWFULLHISTORY_LINK" desc="The label of the Show Full History link at the bottom of the
back/forward menu.">
        Show Full History
```

25 此文件放在目录 src\chrome\app 下，个人觉得放那很诡异。

```
      </message>
    </messages>
  </release>
</grit>
```

说明：

1. Output 节表示输出文件，例如上面的文件会生成一个 generated_resources.h、若干 rc 文件（generated_resources_zh-CN.rc）和若干 pak 文件（generated_resources_zh-CN.pak）。
2. Translations 节表示翻译文件，一般来说每一种支持的语言都应该有一个翻译文件。
3. Messages 节表示定义的默认字符串[26]，

当 grit 解析收到 locate 相关的 grd 文件时，首先生成默认的资源文件，这里默认的资源文件是“en-US”。当发现 Output 节有其他语言的输出时，则查找对应的 xtb 翻译文件，如果 grd 文件中的 message 选项指定需要翻译，则通过 message 中的 name 属性查找 xtb 中对应的 record，然后将替换之。

在 grd 文件中有很多可选的选项，具体可以参考 chrome 自带的 grd 文件或者 grit 源代码。目前本人未找到 google 官方的帮助文档。

Xtb 文件也是一个 xml 文件，典型的内容格式如下：

```
<?xml version="1.0" ?>
<!DOCTYPE translationbundle>
<translationbundle lang="zh-CN">
<translation id="6779164083355903755">删除(&amp;amp;R)</translation>
<translation id="3581034179710640788">此网站的安全证书已过期！</translation>
<translation id="8275038454117074363">导入</translation>
</translationbundle>
```

或者

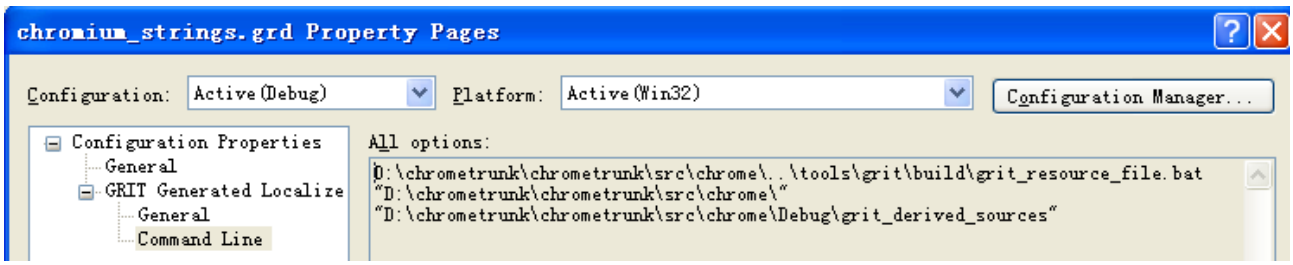
```
<?xml version="1.0" ?>
<!DOCTYPE translationbundle>
<translationbundle lang="zh-CN">
<translation id="IDS_WEB_FONT_FAMILY">Simsun</translation>
<translation id="IDS_WEB_FONT_SIZE">84%</translation>
<translation id="IDS_UI_FONT_FAMILY">default</translation>
</translationbundle>
```

两者区别主要是 ID 的表示方式。后者 ID 和 grd 中的 name 是一致的，而前者则通过某种算法将 grd 中的 name 转换为由数字组成的 ID。

7.4. Grd 文件的编译

chrome 通过添加“自定义编译规则(Custom Build Rules)”来通过 vs2008 自动编译所有自定义格式文件。例如 grd 文件的规则如下：

26 不同的 grd 文件有不同的作用，其他文件的 message 节里面的内容不一定是字符串，也可能是其他类型。



如图所示，grd 使用一个 bat 文件编译，并且提供两个参数。正如前面提到的，第二个参数就是目标文件的输出路径。

该 bat 文件的内容如下：

```
:: Batch file run as build command for .grd files
:: The custom build rule is set to expect (inputfile).h and (inputfile).rc
:: our grd files must generate files with the same basename.
@echo off

setlocal

… 忽略 …

:: Put cygwin in the path
call %SolutionDir%\..\third_party\cygwin\setup_env.bat

%SolutionDir%\..\third_party\python_24\python.exe %SolutionDir%\..\tools\grit\grit.py -i %InFile% build -o %OutDir%
%PreProc1% %PreProc2% %PreProc3% %PreProc4% %PreProc5
```

从上面内容可以发现，chrome 将 python 解释器直接放到源码【src\third_party\python_24】里。然后通过它调用 grit.py 文件实现文件的编译。

关于 grd 的自定义编译规则文件可以查看文件【src\tools\grit\build\grit_resources.rules】。msdn 上【http://msdn.microsoft.com/en-us/library/03t8bzzy.aspx 】也许有些帮助。

7.5. Locale 初始化

当通过 grd 文件生成 locale 下语言项目依赖的文件后，Chrome 将这些项目打包生成一个语言 Dll，这些 Dll 可以在目录【src\chrome\Debug/Release\locales】下找到。当 Chrome 启动时，它会通过某种途径查找 locale 类型，然后找到对应的 Dll 来 load。

Chrome 查找 locale 类型通过【src\chrome\common\l10n_util.h】的 GetApplicationLocale 函数实现。

该函数首先检查命令行有没有通过“--lang”指定 locale，如果没有，则检查当前的配置文件中是否指定 locale。如果未指定，则获取操作系统的 locale，如果再获取失败，则使用默认的 en-US。当然在返回之前，Chrome 总会检查当前的语言文件目录是否存在该语言的 Dll。

chrome 自己弄了一套本地配置系统，下面是一个典型的配置文件格式，内容为语言选项

```
{
   "intl": {
      "app_locale": "zh-CN"
   }
}
```

当找到 locale 后，chrome 通过【src\chrome\common\resource_bundle_win.cc】中的

LoadResources 函数加载对应的 Dll。

7.6. 国际化小结

当程序中需要使用多国语言的字符串时，可以参考下面的代码

```
ResourceBundle &rb = ResourceBundle::GetSharedInstance();
std::wstring str = l10n_util::GetString( IDS_ABCDEFG )
```

其中 IDS_ABCDEFG 就是定义在 grd 中的 name 。

Chrome 中有两个 grd 文件比较重要：

1. src\chrome\app\generated_resources.grd ： 该文件定义了多国语言字符串（string），例如按钮的文字，窗口的标题等。
2. src\chrome\app\resources\locale_settings.grd ： 该文件定义了多语言配置，例如窗口的大小，某些控件的尺寸。考虑到不同语言的字符串差异，同一个窗口在不同语言下要求的宽度等参数可能不一样。此外还包含字体的大小、名称等。

8. UI 主题

UI 主题和国际化一样，同样适用 grit 将 grd 翻译成目标文件，然后再生成主题 Dll。生成主题的 grd 文件是【src\chrome\app\theme\theme_resources.grd】。该文件在项目 chrome_resources 中。该项目生成的 rc 文件被项目 theme_dll 使用并生成一个主题 Dll。

主题 Dll 在目录【src\chrome\Debug/Release\themes】下。在我这个 chrome 源码版本中，只有一个主题 Dll。名字为“default.dll”。这个 Dll 在【src\chrome\common\resource_bundle_win.cc】中的函数 LoadThemeResources()被 load。

主题通常都由一些图片组成，这些图片存放在目录【src\chrome\app\theme】下，基本上时 png 图片格式。

这些图片通过 theme_resources.grd 来生成 dll。这个文件的部分格式如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<grit latest_public_release="0" current_release="1">
  <outputs>
    <output filename="grit/theme_resources.h" type="rc_header">
      <emit emit_type='prepend'></emit>
    </output>
    <output filename="theme_resources.rc" type="rc_all" />
    <output filename="theme_resources.pak" type="data_package" />
  </outputs>
  <release seq="1">
    <includes>
      <include name="IDR_BACK" file="back.png" type="BINDATA" />
      <include name="IDR_BACK_D" file="back_d.png" type="BINDATA" />
    </includes>
  </release>
</grit>
```

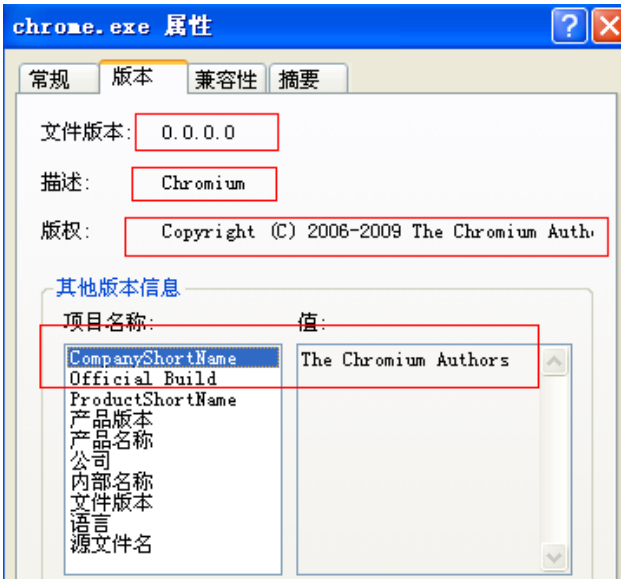
当用户需要使用图片时，可以参考如下代码：

```
ResourceBundle &rb = ResourceBundle::GetSharedInstance();
SkBitmap* img_ = rb.GetBitmapNamed(IDR_ABCDEFG);
```

其中 IDR_ABCDEFG为在 theme_resources.grd中每一张图片分配的name。

9. Chrome 的版本信息

如果查看项目生成的 chrome.exe、chrome.dll 文件的属性，可以发现如下：



实际上这些参数通过项目 chrome_exe、chrome_dll 中的一个.rc.version 文件来获取，之所以讲这些是因为.rc.version 的处理和前面提到的 grd 文件有异曲同工之妙。version 文件典型的内容如下：

```
//////////////////////////////////////////////////////////////////////////////
//
// Version
//
... 省略 ...
        BEGIN
            VALUE "CompanyName", "@COMPANY_FULLNAME@"
            VALUE "FileDescription", "@PRODUCT_FULLNAME@"
            VALUE "FileVersion", "0.0.0.0"
            VALUE "InternalName", "chrome_exe"
            VALUE "LegalCopyright", "@COPYRIGHT@"
... 省略 ...
        END
    END
... 省略 ...
END
```

9.1. version 文件的编译

Version 文件的编译命令如下：

```
D:\chrometrunk\chrometrunk\src\chrome\../chrome/tools/build/win/version.bat
"D:\chrometrunk\chrometrunk\src\chrome\../chrome/"
"D:\chrometrunk\chrometrunk\src\chrome\Debug\obj\chrome_exe"
"D:\chrometrunk\chrometrunk\src\chrome\Debug\obj\chrome_exe/chrome_exe_version.rc"
```

此类型的编译规则文件见【src\chrome\tools\build\win\version.rules】。该规则的核心就是 version.bat 文件。

version.bat 的内容如下：

```
:: Batch file run as build command for vers.vcproj
@echo off

setlocal

set InFile=%~1
set SolutionDir=%~2
set IntDir=%~3
set OutFile=%~4
set VarsBat=%IntDir%/vers-vars.bat

:: Put cygwin in the path
call %SolutionDir%\..\third_party\cygwin\setup_env.bat

:: Load version digits as environment variables
cat %SolutionDir%\VERSION | sed "s/\(.*\)/set \1/" > %VarsBat%

:: Load branding strings as environment variables
set Distribution="chromium"
if "%CHROMIUM_BUILD%" == "_google_chrome" set Distribution="google_chrome"
cat %SolutionDir%app\theme\%Distribution%\BRANDING | sed "s/\(.*\)/set \1/" >> %VarsBat%

set OFFICIAL_BUILD=0
if "%CHROME_BUILD_TYPE%" == "_official" set OFFICIAL_BUILD=1

:: Determine the current repository revision number
set PATH=%~dp0..\..\..\..\third_party\svn;%PATH%
svn.exe info | grep.exe "Revision:" | cut -d" " -f2- | sed "s/\(.*\)/set LASTCHANGE=\1/" >> %VarsBat%
call %VarsBat%

::echo LastChange: %LASTCHANGE%

:: output file
cat %InFile% | sed "s/@MAJOR@/%MAJOR%/" ^
               | sed "s/@MINOR@/%MINOR%/" ^
               | sed "s/@BUILD@/%BUILD%/" ^
               | sed "s/@PATCH@/%PATCH%/" ^
               | sed "s/@COMPANY_FULLNAME@/%COMPANY_FULLNAME%/" ^
               | sed "s/@COMPANY_SHORTNAME@/%COMPANY_SHORTNAME%/" ^
               | sed "s/@PRODUCT_FULLNAME@/%PRODUCT_FULLNAME%/" ^
               | sed "s/@PRODUCT_SHORTNAME@/%PRODUCT_SHORTNAME%/" ^
               | sed "s/@PRODUCT_EXE@/%PRODUCT_EXE%/" ^
               | sed "s/@COPYRIGHT@/%COPYRIGHT%/" ^
               | sed "s/@OFFICIAL_BUILD@/%OFFICIAL_BUILD%/" ^
               | sed "s/@LASTCHANGE@/%LASTCHANGE%/" > %OutFile%

endlocal
```

注意上述红色标示的部分

VERSION 部分是获取 Chrome 预先写好的版本参数文件【src\chrome\VERSION】,内容如下：

```
MAJOR=2
MINOR=0
BUILD=175
PATCH=0
```

BRANDING 部分是获取 Chrome 预先写好的公司参数文件【src\chrome\app\theme\chromium\BRANDING】,内容如下

```
COMPANY_FULLNAME=The Chromium Authors
COMPANY_SHORTNAME=The Chromium Authors
PRODUCT_FULLNAME=Chromium
PRODUCT_SHORTNAME=Chromium
COPYRIGHT=Copyright (C) 2006-2009 The Chromium Authors. All Rights Reserved.
```

如果熟悉 Linux 的话，估计不会对 sed 这条命令陌生。这条命令存在与目录【src\third_party\cygwin】中。sed 命令将上述两个文件的内容经过处理，在每一行之前加入 set。然后输出到%VarsBat%文件【src\chrome\Debug\obj\chrome_exe\vers-vars.bat】中。最终 vers-vars.bat 文件的格式类似【set MAJOR=2】。很显然，当执行这个 bat，即把所有的参数添加的环境变量了。

最后一条命令将 version 文件中【%***%】替换成同名环境变量的值，并输出到一个 rc 文件【src\chrome\Debug\obj\chrome_exe/chrome_exe_version.rc"】中。

最后在 chrome_exe 项目的 rc 文件【src\chrome\app\chrome_exe.rc】中，【chrome_exe_version.rc】被 include 进去。源码如下：

```
#else  // APSTUDIO_INVOKED
#include "chrome_exe_version.rc"
#endif  // APSTUDIO_INVOKED
```