关于midori网页浏览器的界面修改

吕凯

主要内容:

一、简介

二、编译安装

三、源代码分析

四、修改界面举例

一、简介：
==========
midori 是一个轻量级的浏览器,目前已知最新版本是 v0.1.9

http://www.twotoasts.de/index.php?/pages/midori_summary.html
在这里可以找到每个版本的下载。

二、编译安装
=========
2.1 版本选择

下面是一些版本的变化：
v0.0.17:
是目前可以在板子上面编译，以及运行的版本，提供了最基本的浏览器功能。
其源代码的安装过程遵循标准的：
./configure
make
make install

v0.0.18:
增加了全屏显示功能，增加了利用 Gettext 实现本地化语言支持功能。
相对于上个版本，修改了代码的结构，从这个版本之后，编译流程有所变化：
配置：运行 './waf configure'
编译：运行 './waf build'
安装：运行 './waf install'
这里 waf 是一个 python 写的脚本。

v0.1.0:
这个版本允许隐藏菜单栏，
允许定制工具导航栏的风格。

v0.1.3:
这个版本的书签功能有所增强，可以支持树状结构组织书签。
集成了 Maemo 相关的选项

v0.1.5:
这个版本相对于前面，需要更高版本的库，主要库有：
WebKitGTK+ 1.1.1, Glib 2.16 and libsoup 2.25.2

2.2 目前使用的情况：
----
v0.0.17 版本可以在板子和机器上面编译运行，因此可以在这个版本的基础上修改源代码。
后面的版本由于无法编译依赖的库，所以无法编译，没有选择它们。

2.3 确定依赖库的常用的操作
----
1)查看 midori 浏览器运行所依赖的库：
$apt-cache depends midori
输入之后输出：
midori
  依赖：libatk1.0-0
  依赖：libc6
  依赖：libcairo2
  依赖：libfontconfig1
  依赖：libfreetype6
  依赖：libglib2.0-0
  依赖：libgtk2.0-0
  依赖：libpango1.0-0
  依赖：libpixman-1-0

依赖: libpng12-0
依赖: libsexy2
依赖: libwebkitgtk1d
依赖: libx11-6
依赖: libxcursor1
依赖: libxext6
依赖: libxfixes3
依赖: libxi6
依赖: libxinerama1
依赖: libxml2
依赖: libxrandr2
依赖: libxrender1
依赖: zlib1g

2)查看编译，运行 midori 所需要的所有的库依赖:
$apt-cache show midori
这里除了库，还显示了许多其他的信息，例如版本号，以及是否已经安装等等。

3)自动安装编译源代码需要的库(包):
apt-get build-dep packagename
所有的源代码包有一个'Build-Depends'域，指明了从源代码编译这个包需要安装的软件。上面的命令可以下载这些包，其中'packagename'是包名。

4)运行时自动安装缺失的软件:
auto-apt run command
这里，command 是你要运行的命令。例如 auto-apt run ./configure .当缺失依赖软件的时候，它会询问你安装，方便了编译。安装之后会继续运行。auto-apt 需要保持一个最新的数据库，利用这个命令进行更新: auto-apt update, auto-apt updatedb and auto-apt update-local。

5)下载源代码,如果可以，就把源代码自动编译成.deb 包:
$ apt-get -b source packagename
这里-b 选项指定要把源代码编译成.deb 包，如果不需要，则可去掉-b.

*三、源代码分析
==========
这里主要针对 ui 修改部分进行了分析。

3.1 程序运行的整体过程可以参见:
src/main.c 中
---
main 函数:
主要有三个步骤: 初始化配置, 启动浏览器, 保存配置退出。

1)初始化配置:
```
gchar* configPath = g_build_filename(g_get_user_config_dir(), PACKAGE_NAME, N    ULL);
```
这句话确定了配置文件的位置,默认为~/.config/midori, 在这里之后的内容都和初始化相关。

2)启动浏览器:
```
browser = browser_new(browser);
```
这句话建立浏览器, 并启动。

3) 保存配置退出:
这个动作发生在你关闭浏览起的时候。
gtk_main();之后的语句对应于保存你在浏览器会话期间进行的配置等的保存, 保存之后就退出了。

3.2 浏览器界面相关文件

对应浏览起界面的修改, 主要工作集中在第 2 个过程中的 browser_new 中。
browser_new 定义在 src/browser.c 中, 这里面根据浏览器的界面定义相关信息, 添加了浏览器所需的所有构件界面。

浏览器的界面相关信息部分, 主要在如下文件中体现:
global.h:
这个文件定义了一些全局的信息, 主要是浏览器所使用的图标 id

ui.h:
这个文件定义了一个字符数组, 该数组以类似 xml 格式的方式指明了浏览器界面中包含的每个构件, 以及它们之间的关系。

browser.h
这个文件定义了建立浏览器构件需要的一些定义信息, 最主要包含了浏览器中每个构件相关联的动作, 并且为这些动作连接相应的处理 (回调) 函数。

browser.c
这里包含了所有界面的实现以及界面相应动作的实现。

3.3 浏览起界面的建立大体过程

在 browser_new 中, 建立界面的过程大体是这样的:
1) 根据 browser.h 相关定义, 得到浏览器每个构件的动作信息。
这里涉及到的 gtk 相关构件类型: GtkActionGroup,GtkActionEntry,GtkAction
2) 根据 ui.h 相关定义, 得到浏览器每个构件的层次结构以及组成等界面信息。
这里涉及到的 gtk 相关构件类型: GtkUIManager
3) 根据前面得到的信息, 建立浏览器以及每个构件。
4) 根据配置或者其他需要, 获取某些建立好的子构件, 设置它们的特殊属性 (例如设置某个按钮的显示图标,

而图标信息在global.h中有定义）

四、修改界面举例:
==========

*去掉菜单的方法:
 方法1, 把ui.h中有关菜单的地方去掉。
但是这样有警告错误。
方法2, 修改browser.c中的new函数。
在gtk_widget_show(browser->menubar);的后面加上一句:
gtk_widget_hide(browser->menubar);//1vkaiadd

*把工具条变到底下的方法:
修改browser.c中的new函数, 如下:
//gtk_box_pack_start(GTK_BOX(vbox), browser->navibar, FALSE, FALSE, 0);//1vk aide1
gtk_box_pack_end(GTK_BOX(vbox), browser->navibar, FALSE, FALSE, 0);//1vkaiad d

*去掉状态栏和搜索栏的方法:
修改browser.c中的new函数, 如下:
去掉状态栏,注释掉如下的语句:
gtk_box_pack_start(GTK_BOX(vbox), browser->statusbar, FALSE, FALSE, 0);
去掉搜索栏则把如下这句话注释掉:
gtk_toolbar_insert(GTK_TOOLBAR(browser->navibar), toolitem, -1);

*添加一个书签按钮到工具栏的方法:
修改browser.c中的new函数, 如下:
在sokoke_container_show_children(GTK_CONTAINER(browser->navibar));的前面找个位置添加如下代码:
```
toolitem = gtk_tool_button_new_from_stock (GTK_STOCK_EDIT);
g_signal_connect(G_OBJECT(toolitem), "clicked",
                    G_CALLBACK(my_pop_menu), browser->menu_bookmarks);
gtk_toolbar_insert(GTK_TOOLBAR(browser->navibar), toolitem, -1);
```
其中my_pop_menu是回调函数, 用于弹出菜单, 如下:
```
void my_pop_menu(GtkWidget *button, gpointer data)
{
           GtkMenu *menu = GTK_MENU(data);
           g_print("pop\n");
           gtk_menu_popup(menu, NULL, NULL, NULL, NULL,
                          0, gtk_get_current_event_time());
}
```

*添加书签菜单的另外一个方法(利用GtkActionEntry和GtkUIManager):
1)browser.h中添加如下:
```
void
my_bookmark_pop_menu(GtkAction*, CBrowser*);//1vkaiadd
```
2)browser.h中entries[]添加如下一个元素:
```
 { "MyBookMark", MY_STOCK_BOOKMARK//1vkaiadd,for bookmark
         , NULL, ""
                  , "popup my bookmark", G_CALLBACK(my_bookmark_pop_menu) },
```
3)ui.h中的ui_markup中的"<toolbar name='toolbar_navigation'>"部分添加如下:
```
"<toolitem action='MyBookMark'/>"//1vkaiadd
```
4)browser.c中实现一个如下的回调函数:
```
void my_bookmark_pop_menu(GtkAction* action, CBrowser* browser)
{
       GtkMenu *menu = browser->menu_bookmarks;
```

```
        g_print("pop\n");
        gtk_menu_popup(menu, NULL, NULL, NULL, NULL,
                       0, gtk_get_current_event_time());
}
```

*更改工具栏背景图片:
修改 browser.c 中的 new 函数, 如下:
添加这句话:
gtk_rc_parse("myrc");

然后在当前目录下建立相应的资源文件, 定义资源风格, 最重要的是资源文件中这句话:
widget_class "GtkWindow" style "toolbar"

*更改工具栏上面按钮的图标:
方法比较复杂, 需要使用自定义的图标, 自定义一个初始化函数等, 涉及到的文件和函数分别如下:
1)main.c
定义初始化相关的信息
定义 my_custom_icon 初始化自定义图标的函数并 在 main 函数中添加,my_custom_icon();
2)global.h
在这里面定义自己定义的图标的 id 如下:

```
#define MY_STOCK_BACK "qgn_browser_back.png"
#define MY_STOCK_FORWARD "qgn_browser_next.png"
#define MY_STOCK_REFRESH "qgn_browser_refresh.png"
#define MY_STOCK_DELETETAB "qgn_browser_delete.png"
#define MY_STOCK_STOP "qgn_browser_stop.png"
#define MY_STOCK_BOOKMARK "qgn_browser_bookmark.png"
```

3)browser.h
如下的数组中定义了在哪里使用这些图标, 只需要在相应的地方做修改即可。
static const GtkActionEntry entries[]