qtwebengine编译调试定位方法



统信软件技术有限公司

**目录**

[1 Qt WebEngine概述 3](#_Toc390417205)

[2 Qt WebEngine模块介绍 4](#_Toc1937634500)

[2.1 Qt WebEngine Core模块 4](#_Toc1324278682)

[2.2 Qt WebEngine Widgets模块 4](#_Toc1243885336)

[2.3 Qt WebEngine模块 4](#_Toc2053579122)

[2.4 Qt WebEngine Process功能模块 5](#_Toc453141689)

[2.5 开发Qt WebEngine Widgets应用 5](#_Toc597308349)

[2.6 开发Qt WebEngine相关Qt Quick应用 6](#_Toc1729949589)

[2.7 使用Qt WebEngine Core模块 7](#_Toc24876720)

[2.8 Qt WebEngine调试和分析 7](#_Toc1294177860)

[2.8.1 开发人员工具 7](#_Toc831706053)

[2.8.2 命令行参数 8](#_Toc1881929246)

[3 Qt WebEngine其它说明 9](#_Toc1263263185)

[4 编译方法 10](#_Toc1248984606)

[4.1 debian命令编译打包 10](#_Toc1318139751)

[4.1.1使用debuild编译打包 10](#_Toc116974470)

[4.1.2使用dpkg-buildpackage编译打包 11](#_Toc45521590)

[4.1.3使用qmake编译 11](#_Toc460999181)

[4.1.4 编译指定版本qtwebengine 12](#_Toc548167551)

[4.1.5 编译带调试信息的动态库 13](#_Toc1041066308)

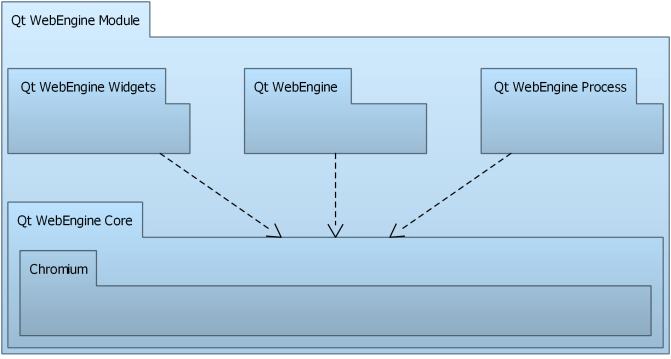
[5 测试功能 14](#_Toc903543813)

[6 参考资料 14](#_Toc565468788)

# 1 WebEngine概述

Qt WebEngine模块提供了一个Web浏览器引擎，可以轻松地将网络内容嵌入到没有本机Web引擎平台Qt应用中。Qt WebEngine使用C++和QML用于渲染HTML、XHTML和SVG类型文档，支持使用级联样式表（CSS）样式设置，和JavaScript编写的脚本。功能被拆分为3个模块和一个独立可执行程序模块。

| **模块名称** | **描述** |
| --- | --- |
| Qt WebEngine Core（核心模块） | 用于与Chromium进行交互 |
| Qt WebEngine Widgets（小部件模块） | 用于创建基于窗口小部件的Web应用程序 |
| Qt WebEngine（Quick模块） | 用于创建基于Qt Quick的Web应用程序 |



Qt WebEngine Process功能负责页面渲染和JavaScript执行，独立于GUI界面进程。如果将Qt库捆绑到应用程序中，则该可执行文件必须随应用程序一起提供

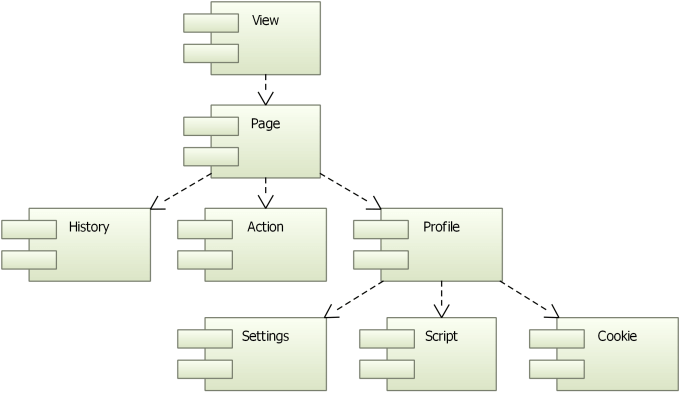
# 2 WebEngine模块介绍

## 2.1 WebEngine Core模块

Qt WebEngine核心基于Chromium项目,Chromium底层相关子模块紧密联系提供了自己的网络和绘画引擎。Qt WebEngine虽然基于Chromium，但不包含或使用任何Chromium提供的服务或附加组件，这些服务或附加组件可能是由Google构建所属Chrome浏览器的一部分。

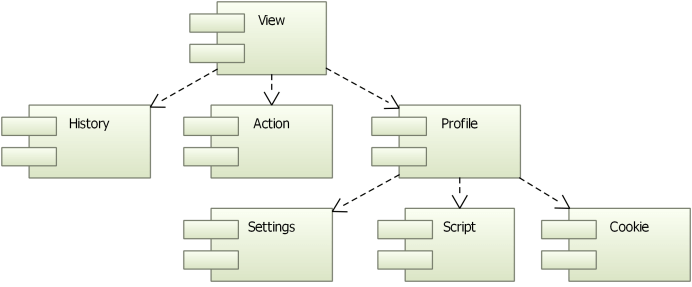
## 2.2 WebEngine Widgets模块

Web引擎视图是Qt WebEngine模块的主要部件组件。它可以用于各种应用程序中加载Web内容。在视图中Web引擎页面包含一个主框架，该主框架负责处理Web内容、导航链接的历史记录和常规处理操作。视图和页面非常相似，它们提供了一组通用功能。所有页面均属于Web引擎配置文件，其中包含共享设置、脚本和cookie。配置文件可用于将页面彼此隔离。典型用例是隐身浏览模式配置文件，不会永久保存任何信息。Qt WebEngine Widgets模块使用Qt Quick场景图将网页元素组成一个视图，所以UI界面进程需要OpenGL ES 2.0或OpenGL 2.0进行渲染。



## 2.3 WebEngine模块

Qt WebEngine主要内容是QML功能实现,包含与Qt WebEngine Widgets实现相同元素，不同之处是没有可单独访问Web引擎页面。支持的页面功能被集成到Web引擎视图中。



## 2.4 WebEngine Process功能模块

Qt WebEngine Process是一个单独的可执行文件，用于呈现网页和执行JavaScript。这样可以缓解安全问题，同时隔离特定内容处理显示可能产生的异常而引起的崩溃。

## 2.5 开发WebEngine Widgets应用

使用QWebEngineView类以最简单方式显示网页，它是一个小部件，所以可以将QWebEngineView对象嵌入到窗体中，实现下载并显示网站。

QWebEngineView \*view = new QWebEngineView(parent);

view->load(QUrl("https://www.chinauos.com"));

view->show();

QWebEngineView的每个实例对象都有一个QWebEnginePage成员对象，QWebEnginePage可以拥有QWebEngineHistory（可访问页面导航历史记录）和几个QAction对象，这些对象在网页上提供功能操作接口。同时QWebEnginePage能够在页面主框架上下文中运行JavaScript代码，并能够为特定事件（例如显示自定义身份验证对话框）启用自定义处理程序。每个QWebEnginePage属于QWebEngineProfile，可以有一个QWebEngineSettings对象处理页面设置，还有一个QWebEngineScriptCollection类对象负责运行页面上的脚本，以及QWebEngineCookieStore对象类负责访问Chromium中HTTP的cookies。

对于基于窗口部件Widgets应用程序在启动时自动初始化Web引擎，在一种情况下除外，将Qt WebEngine集成到插件中，在这种情况下，必须使用QtWebEngine::initialize在应用程序main中对其进行初始化，如以下代码片段所示:

int main(int argc, char \*\*argv){

QApplication app(argc, argv);

QtWebEngine::initialize();

QMainWindow window;

window.show();

return app.exec();

}

## 2.6 开发WebEngine相关Qt Quick应用

WebEngineView的QML类型允许Qt Quick应用程序呈现动态网页内容，WebEngineView可以与其他类型QML或作为Qt Quick应用程序指定的整个screen共享screen。

为了确保可以在GUI显示和渲染过程之间共享OpenGL上下文，必须在应用程序main中调用QtWebEngine::initialize初始化Web引擎，如以下代码段所示：

int main(int argc, char \*argv[]){

QGuiApplication app(argc, argv);

QtWebEngine::initialize();

QQmlApplicationEngine engine;

engine.load(QUrl("qrc:/main.qml"));

return app.exec();

}

应用程序通过使用URL或HTML字符串把页面加载到WebEngineView中，在会话历史记录中可以导航。默认情况下，指向不同页面的链接会加载到同一个WebEngineView对象中，有的网站可能会要求将新页面打开为新的选项卡，窗口或对话框。以下示例QML应用程序使用url属性加载网页：

import QtQuick 2.0

import QtQuick.Window 2.0

import QtWebEngine 1.0

Window {

width: 1024

height: 750

visible: true

WebEngineView {

anchors.fill: parent

url: "https://www.chinauos.com"

}

}

## 2.7 使用WebEngine Core模块

Qt WebEngine Core提供了Qt WebEngine和Qt WebEngine Widgets共享的API，用于处理向Chromium网络堆栈发出的URL请求以及访问其HTTP cookie。自定义实现QWebEngineUrlRequestInterceptor接口并将拦截器安装配置好，可以实现在URL请求（ChWebEngineUrlRequestInfo）到达Chromium网络堆栈之前，对其进行拦截、阻止和修改。QWebEngineUrlSchemeHandler可以注册为配置添加自定义URL方案请求,然后把此自定义URL方案请求作为QWebEngineUrlRequestJob对象传递给QWebEngineUrlSchemeHandler::requestStarted()。QWebEngineCookieStore类提供的功能访问的HTTP cookies，这些功能可用于将Cookie与QNetworkAccessManager同步，以及在导航期间设置、删除和拦截Cookie。

## 2.8 WebEngine调试和分析

### 2.8.1 开发人员工具

Qt WebEngine模块提供了Web开发人员工具，可轻松检查和调试任何Web内容的布局和性能问题。使用基于Chromium或Qt WebEngine的浏览器（例如Chrome浏览器），可以将开发人员工具作为本地网页进行访问。要激活开发人员工具，请启动将Qt WebEngine与命令行参数一起使用的应用程序：

--remote-debugging-port = <port\_number>

其中<port\_number>是指本地网络端口，可以通过在该地址启动浏览器来访问Web开发人员工具http://localhost:<port\_number>，比如在UOS上调试执行qtwebengine示例minimal在浏览器中输入http://127.0.0.1:8000 即可访问工具，运行示例如下

$ ./minimal --remote-debugging-port=8000

DevTools listening on ws://127.0.0.1:8000/devtools/browser/9010ea16-770f-414b-932b-910253ee0ae5

Remote debugging server started successfully. Try pointing a Chromium-based browser to http://127.0.0.1:8000

或者，可以设置环境变量QTWEBENGINE\_REMOTE\_DEBUGGING，可以将其设置为只是一个类似于端口的端口，执行效果如下

$ QTWEBENGINE\_REMOTE\_DEBUGGING=8000 ./minimal

DevTools listening on ws://127.0.0.1:8000/devtools/browser/3a07b8d6-fbd8-4630-91cf-f9baef228cbc

Remote debugging server started successfully. Try pointing a Chromium-based browser to http://127.0.0.1:8000

### 2.8.2 命令行参数

调试时可以使用以下命令行参数作为错误报告输入：

* **--disable-gpu** 禁用GPU硬件加速。在诊断OpenGL问题时，这很有用。
* **--disable-logging** 禁用控制台日志记录，这可能对调试版本很有用。
* **--enable-logging --log-level=0** 启用控制台日志记录并将日志记录级别设置为0，这意味着严重性info及更高级别的消息将记录在日志中。这是调试版本的默认设置。其他可能的日志级别是1警告，2错误和3致命错误。
* --v=1 将日志记录级别提高到最大--log-level，并允许将调试消息记录到详细级别1。较高的数字进一步增加了详细程度，但可能导致大量已记录的消息。默认值为0（无调试消息）。
* **--no-sandbox** 禁用渲染器和插件进程的沙箱。请记住禁用沙箱可能会带来安全风险。
* **--single-process** 在与浏览器相同的过程中运行渲染器和插件。这对于获取渲染器崩溃的堆栈跟踪很有用。
* **--enable-features=NetworkServiceInProcess** 在主要过程中运行网络。这可能有助于防火墙管理，因为仅需要将应用程序可执行文件列入白名单，而不需要考虑QtWebEngineProcess，但这意味着失去网络服务沙箱安全性。

或者，可以设置环境变量QTWEBENGINE\_CHROMIUM\_FLAGS。例如可以将以下值设置为在调试名为simplebrowser的应用程序时禁用日志记录：

$ QTWEBENGINE\_CHROMIUM\_FLAGS = “ --disable-logging” simplebrowser

如果可以确保应用进程调用QtWebEngine::initialize()之前设置QTWEBENGINE\_CHROMIUM\_FLAGS，也可以在应用程序中使用qputenv函数来设置QTWEBENGINE\_CHROMIUM\_FLAGS添加调试参数。

# 3 WebEngine其它说明

Qt WebEngine当前仅支持Windows、Linux和macOS，由于Chromium编译构建要求，通常还需要比Qt其它部分依赖更新的编译器。

Qt WebEngine取代了Qt WebKit模块，该模块基于WebKit项目，自Qt5.2以来已停止与上游WebKit进行主动同步，并在Qt5.5中已经弃用。在Qt的WebView模块允许使用的平台上的本地Web浏览器，其中一个是可用的。

Qt WebEngine使用其自己的网络堆栈，QSslConfiguration被设计目的并非是为了打开SSL连接,而是日通过Qt WebEngine使用操作系统中CA根证书来验证对等方证书。WebEngineCertificateError::error和QWebEngineCertificateError::error枚举提供有关各类可能出现证书错误的信息。可以使用WebEngineView::certificateError QML方法或重新实现函数QWebEnginePage::certificateError来处理错误。

Qt WebEngine渲染Web页面需要OpenGL支持，OpenGL和硬件密切关联，可以通过字符查看GPU支持情况，以下示例展示GPU详细信息:

int main(int argc, char \*argv[]){

QCoreApplication::setAttribute(Qt::AA\_EnableHighDpiScaling);

QApplication app(argc, argv);

QWebEngineView view;

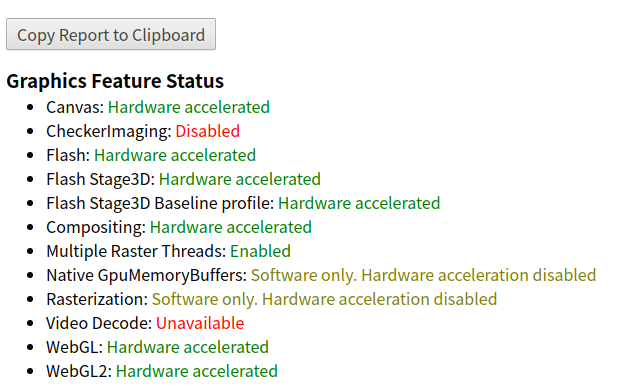
view.setUrl(QUrl(("chrome://gpu"));

view.resize(1024, 750);

view.show();

return app.exec();

}

运行结果如下图intel i5-6300U非独立显卡平台信息  


调试OpenGL相关功能可以借助qtbase中自带demo可以测试和验证[examples/opengl]qt官方gerrit地址如https://codereview.qt-project.org/gitweb?p=qt/qtbase.git;a=tree;f=examples/opengl;h=a96feac5d43027795ff0af6ed1a8cdcf1a6a9ae2;hb=refs/heads/5.15.0

# 4 编译webengine

## 4.1使用debuild编译打包

编译打包之前需要安装依赖,执行如下命令

$ sudo apt build-dep .

需要安装开发工具包devscripts命令如下

$ sudo apt install devscripts

$ debuild -b

## 4.2 使用qmake编译

qtwebengine版本和qt版本差异建议不要太大,当前系统中qt版本和需要编译qtwebengine版本差异较大时,先编译qteverywhare，然后再单独编译qtwebengine，以在系统环境qt5.11.3上编译5.15.1为例，编译前从qteverywhere中删除qtwebegnine的目录

为编译qt-everywhere设置指定安装目录

暂且设置安装到目录/opt/qt5.15.1\_everywhere,编译前的配置

$ ./configure -opensource -confirm-license -prefix /opt/qt5.15.1\_everywhere -release -opensource -plugin-sql-mysql -plugin-sql-sqlite -no-sql-sqlite2 -plugin-sql-tds -system-sqlite -system-harfbuzz -system-zlib -system-libpng -system-libjpeg -system-doubleconversion -system-pcre -openssl -no-rpath -verbose -optimized-qmake -dbus-linked -no-strip -no-separate-debug-info -qpa wayland -xcb -glib -icu -accessibility -no-directfb -no-use-gold-linker -recheck-all -nomake examples

编译并安装qteverywhere

$ make -j `nproc`

$ make install

**编译qtwebengine模块**

设置环境变量，使得qtwebengine编译使用qt5.15.1的环境

export QT\_BUILDDIR=/opt/qt5.15.1\_everywhere

export LD\_LIBRARY\_PATH=$QT\_BUILDDIR/lib:$LD\_LIBRARY\_PATH

export QT\_QPA\_PLATFORM\_PLUGIN\_PATH=$QT\_BUILDDIR/plugins/platforms

export QT\_PLUGIN\_PATH=$QT\_BUILDDIR/plugin

export PKG\_CONFIG\_PATH=$QT\_BUILDDIR/lib/pkgconfig:$PKG\_CONFIG\_PATH

export PATH=$QT\_BUILDDIR/bin:$PATH

qtwebengine从debian上游拉取的源码，执行命令应用一下patch

$ dh\_quilt\_patch

安装依赖库

$ sudo apt build-dep .

编译webengine前的配置

$ qmake .. QT\_BUILD\_PARTS+=tests QMAKE\_EXTRA\_ARGS+=" -proprietary-codecs -system-webp -qt-webengine-icu"

编译并安装

$ make -j `proc`

$ make install

## 4.3 编译带调试信息的动态库

正常打包编译做gdb调试的时候，无法深入跟进chromium代码，需要编译带调试符号动态库，只需要修改文件qtwebengine.pro在文件第一行添加如下代码即可，编译出来的动态库可以接近200MB以上包含一定符号信息。

CONFIG += debug

# 5 测试功能

编译自带example示例简单浏览器,测试正常可以浏览网页即表示功能ok  
源码路径 ./qtwebengine-opensource-src/examples/webenginewidgets/simplebrowser

# 6 参考资料

Qt WebEngine Overview: [https://doc.qt.io/qt-5/qtwebengine-overview.html](https://doc.qt.io/qt-5/qtwebengine-overview.html" \o "https://doc.qt.io/qt-5/qtwebengine-overview.html)