node-webkit：HTML5桌面应用运行环境

文 / 王文睿

node-webkit是一个支持用HTML5技术编写桌面应用的运行环境。它集成了Node.js和Chromium中的HTML5引擎，使开发者既可以用HTML、CSS和JavaScript编写用户界面，又可以直接调用Node.js平台上的众多库模块，拥有访问本地操作系统的能力。

提到node-webkit，就不得不先说说Node.js，它提供了用JavaScript语言编写本地应用程序的运行环境。使用Node.js，JavaScript代码能脱离浏览器，以脚本形式运行在本地操作系统中。由于极大地扩展了JavaScript程序的能力，Node.js平台上涌现了大量程序库和应用。它们都收录在npm这个仓库中（npmjs.org）。最近两年，npmjs.org上的软件包数量已从4000多个增长到5万多个。

node-webkit的想法是把Node.js和浏览器的功能结合在一起，在两种环境间提供互通访问。目前，node-webkit支持Windows、OS X和Linux三种主流的桌面系统平台。在它之上诞生了包括集成开发环境、游戏、工具、终端和企业等各类应用。在过去一年中，node-webkit项目发布了20个新版本（其中包含5个主要版本）。它在GitHub上有7000多人关注，在所有C++项目中排行第三。

为什么要使用node-webkit

2011年底，node-webkit项目诞生于英特尔开源技术中心，当时它的创建者正在寻找一种为Web Runtime项目扩充API的途径时有了这个想法：在HTML DOM环境中直接调用Node.js平台上的库。这样既可在编写HTML5界面应用时有大量本地API使用，不再受浏览器环境的限制；从另一方面来看，对于那些已熟悉Node.js的开发者来说，node-webkit也提供了Node.js程序的GUI支持。举个简单的例子，这个想法使得下面的代码成为可能，它在HTML页面中使用“fs”模块列出当前目录的内容。

```
<html><head>
    <title>testfs</title>
    <script>
    var fs = require('fs');

    function test_fs() {
      var output = document.
getElementById ('output');
      output.innerHTML = '';
      fs.readdir(".", function (err,
files) {
        var result = '';
        files.forEach(function
(filename) { result += filename +
'<br/>'; }
      );
        output.innerHTML = result;
      });
    }
    </script></head>
    <body onload="test_fs()">
    <p id="output"></p>
    </body>
    </html>
```

另外，node-webkit提供了将HTML、CSS和JavaScript代码打包成本地可执行程序的方法。对最终用户来说，使用node-webkit编写的程序看起来和传统本地应用几乎没什么区别，而且有着更绚丽的界面。例如，Sputnik这个基于node-webkit的RSS阅读器被使用者评论为“运行它，你不会意识到它是用Web技术编写的，它看起来就是一个很不错的本地应用，而且在你的Mac和Windows PC上都能运行”。

得益于HTML5技术的跨平台特性，node-webkit应用可以很轻松地做到跨平台支持，绝大多数代码都可以在平台之间共享。大多数情况下，开发者要做的只是将代码重新打包并测试就可以了。

node-webkit内部架构

node-webkit基于Chromium项目内部的content模块。Google Chrome浏览器也是基于同样的代码。

如图1所示，Chrome浏览器的大部分功能都被封装在content模块以下的部分，包括WebKit（现在叫Blink）、网络、V8 JavaScript引擎等。名为“chrome”的方框则是Chrome浏览器特别实现的部分，例如窗口和菜单界面、用户Profile管理等。

node-webkit和“chrome”处于同样的位置，这意味着它是一个特别的浏览器，去除Chrome中不必要的部分，只保留加载和运行页面必要的功能。

Chromium和Node.js的集成

node-webkit架构中另外一个重要的部分是Chromium（content模块）和Node.js的集成方式。这首先需要了解Chromium架构的另一个重要方面：多进程模型。

如图2所示，在Chromium的多进程模型中有两类主要进程：Browser和Renderer进程。负责网页渲染和JavaScript执行的WebKit等都位于Renderer进程中；而Browser进程则负责所有本地平台相关的功能，例如窗口创建、文件存取和网络访问等。当Renderer进程加载或渲染网页时，它就通过Browser进程完成所需的网络、文件访问等操作。二者之间通过一个高效的IPC通道通信（当然，渲染图像等大块数据通过其他更快的通道完成）。

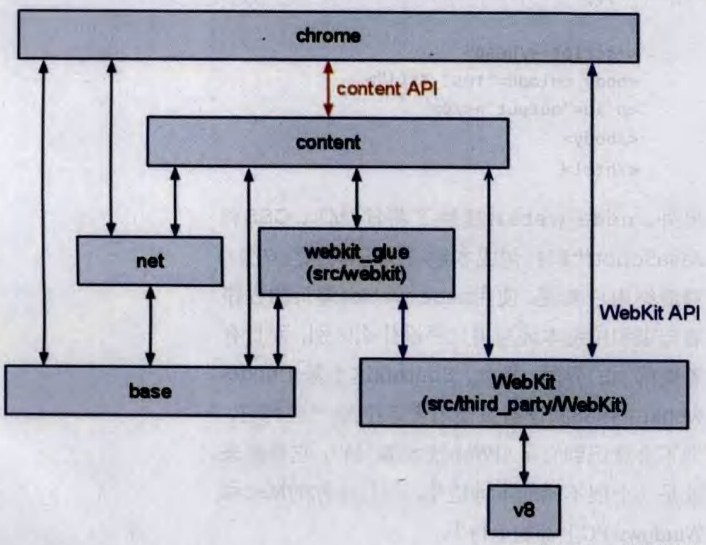

图1 Chrome浏览器架构

node-webkit的一个重要设计目标就是提供DOM和Node.js之间的直接和高效访问。这意味着二者环境中的JavaScript对象能直接互相引用，彼此之间的函数调用也可以直接完成。因此，Node.js被集成在Renderer进程中，并和WebKit同处于同一线程（Renderthread）中。

与WebKit一样，Node.js内部也是基于事件循环驱动的结构。要将二者集成在一个线程中，必须将二者的事件循环合并在一起。Node.js的事件循环由libuv提供，而WebKit的事件循环只是简单地接收、发送和Browser进程之间的IPC消息。因此，在node-webkit中使用libuv重新实现了和Browser进程的IPC通信（见MessagePumpUV）。

另外，完成Node.js和DOM之间JavaScript互通的目标当然还需要双方的JavaScript对象都处于同一个V8引擎中。因此，在node-webkit中，来自Node的V8引擎和Chromium中的V8引擎被合并，结果是使用Chromium中的版本，且Node中所需的JavaScript环境（Context）由Chromium创建并交由后者使用。

消息循环及V8引擎的合并是node-webkit项目早期的核心工作。为了在三个目标操作系统上同时完成这一目标，项目的架构从WebKitGtk迁移到了同样基于Chromium的CEF。

node-webkit的内置API

除了HTML5和Node.js平台所提供的功能外，node-webkit还需要向开发者提供操作本地UI的能力。这部分API被包装在一个内置的叫做“nw.gui”的库中。它提供的功能包括：创建窗口、本地菜单、系统托盘、Shell交互、剪贴板和文件对话框等。具体的功能列表和API文档请参考项目WiKi。

这部分工作要感谢项目的另一位主要贡献者赵成为node-webkit实习半年期间所做的工作。同样遵循了Chromium的多进程架构，每个API的实现部

分位于Browser进程中，对应的JavaScript对象位于Renderer进程中。其中定义了一套协议来维护两边对象的对应关系。在实现这部分功能时，为了实现和维护的便利，整个项目的架构从CEF迁移到了Chromium Content API。

不同的安全模型

node-webkit代码和浏览器中运行代码的不同，除Node.js支持之外，另一主要区别就是不同的安全模型了。浏览器运行的网站代码有如下特点。

- 在用户上网时，从互联网下载并立即执行。
- 默认情况下不受信任。
- 运行时访问特殊功能需要用户批准。

而node-webkit支持的是本地桌面应用，类似读者常用的Outlook或Skype，它们对应的区别如下。

- 用户从官方渠道获得安装文件并安装。
- 默认情况下受到信任。
- 软件几乎可以做任何事情。

一旦处于不同的安全模型下，很多在浏览器环境中不被允许的操作应该被支持。因此，在兼容标准HTML5的基础之外，node-webkit提供了额外的功能：跨域访问——Node.js的代码可以访问任意域的DOM；DOM中的文件操作——JavaScript代码可设置Input标签的值。

node-webkit的典型应用

简而言之，node-webkit支持用Web技术开发本地桌面应用。在互联网大潮的推动下，HTML5已是构建用户界面的最佳选择，它一定会被广泛使用在桌面应用程序中。

在node-webkit开发的初始阶段，便吸引了一些有名的用户。Light Table是一个创新的集成开发环境，该项目就基于node-webkit构建。在获得Y Combinator投资之前曾在KickStarter网站上募集到来自7千多人的35万美元投资。在被问到为什么使用Web技术和node-webkit开发时，创始人Chris Granger说“原因很简单，HTML有难以置信的灵活性，能进行快速开发，每天可以加入很多功能；业界的大量投入会让这项技术变得越来越好”。

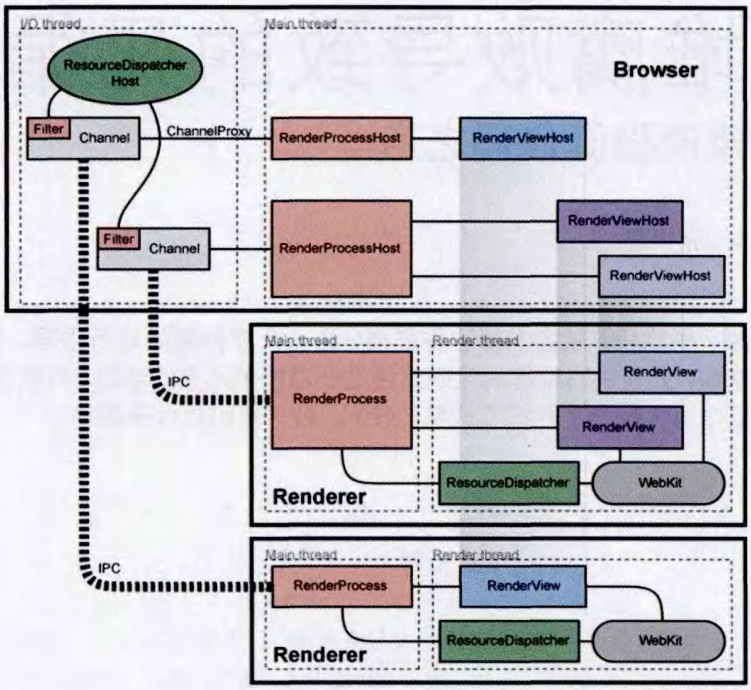

图2 Chromium的多进程模型

Leap Motion是一家设计和生产高精度体感控制器的公司，他们的控制器能让用户以一种创新的方式同电脑交互。能有大量利用这种交互方式的应用程序是成功的关键。为了鼓励开发者，Leap Motion推出了应用程序市场并支持用JavaScript操纵Leap Motion控制器。他们的应用市场客户端AirSpace Home便是用node-webkit开发的。Node.js支持能让他们方便地访问底层设备。

使用node-webkit开发的游戏也有很多。node-webkit可帮助游戏开发者跨平台，另外，包装成App方式也使得游戏更容易商业化。Game Dev Tycoon是基于node-webkit的模拟经营类游戏，该游戏目前已发布到SteamWorks平台。它和SteamWorks平台集成的部分是一个Node.js模块，目前已开源，这会帮助更多node-webkit游戏与SteamWorks集成。

王文睿

node-webkit项目创建和维护者，任职于英特尔开源技术中心。曾从事浏览器和Web技术、MeeGo、Tizen、JNI XML性能库的开发和数据中心解决方案咨询业务。毕业于中国科学技术大学计算机系，研究方向为高性能计算。

责任编辑：卢鹄翔（ludx@csdn.net）