申请上海交通大学工程硕士学位论文

基于Gstreamer的Chromium音视频播放系统的设计与实现

|  |  |
| --- | --- |
| 学校代码： | 10248 |
| 作者姓名： | 胡济豪 |
| 学 号： | 1130379118 |
| 第一导师： | 姚建国 |
| 第二导师： |  |
| 学科专业： | 软件工程 |
| 答辩日期： | 年 月 日 |

上海交通大学软件学院

2016年 11 月

A Dissertation Submitted to Shanghai Jiao Tong University

for Master Degree of Engineering

**THE DESIGN AND IMPLEMENTATION**

**OF CHROMIUM AUDIO AND VIDEO PLAYER**

**BASED ON GSTREAMER**

|  |  |
| --- | --- |
| University Code： | 10248 |
| Author： | HuJihao |
| Student ID: | 1130379118 |
| Mentor 1： | YaoJianguo |
| Mentor 2: |  |
| Field： | Software Engineering |
| Date of Oral Defense： |  |

School of Software

Shanghai Jiaotong University

Nov. 2016

**上海交通大学**

**学位论文原创性声明**

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期： 年 月 日

**上海交通大学**

**学位论文版权使用授权书**

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

**保密**□，在 年解密后适用本授权书。

本学位论文属于

**不保密**🗹。

（请在以上方框内打“**√**”）

学位论文作者签名： 指导教师签名：

日期： 年 月 日 日期： 年 月 日

基于Gstreamer的Chromium音视频播放系统的设计与实现

摘 要

随着汽车的普及和生活节奏的加快，人们在日常生活中越来越多的使用汽车。对于车载多媒体系统，广大消费者已经不再仅仅满足于本地音视频的播放，更希望在车载终端上随时随地享受互联网多媒体服务，例如打开车载终端的浏览器直接联网浏览网页，播放喜欢的音视频，或打开音乐电台在线聆听最新音乐。本课题希望通过对嵌入式多媒体应用的现状和趋势做分析，结合车载多媒体终端软硬件开发平台技术，对常见构建多媒体应用的开源框架及浏览器音视频播放体系结构的相关技术进行学习研究，完成基于Gstreamer的Chromium浏览器音视频播放系统的设计与实现。

本文的主要工作包括：

（1）通过对当前主流浏览器进行了解，概括其基本特性。系统性地阐述浏览器将HTML、CSS、JavaScript文本文档转变成可视化图像界面的基本流程和工作原理。在此前提下，以渲染引擎WebKit为例介绍其内部设计，并对以Webkit为内核的浏览器Chromium做整体框架分析。

（2）以Chromium浏览器多媒体播放为重点，详细分析相关模块的设计和实现细节，对涉及到的类和流程做详细的说明与解析，了解并掌握其内部实现机制。根据产品需求，对现存的框架进行重构与优化。

（3）深入研究Gstreamer的多媒体框架和重要组成部分，详细阐述Gstreamer开发的核心概念和编程基础，简要介绍GStreamer的系统集成方法、调试方法、以及调用机制。基于Gstreamer框架，设计和实现音视频文件的播放，并对实现细节进行封装，以便第三方使用和二次开发。

（4）基于对Gstreamer多媒体播放底层的接口封装和Chromium多媒体部分的重构，重新设计和实现中间层模块，将Chromium与Gstreamer进行对接。

（5）测试和验证设计构想和实现方案，对存在的不足进行讨论，提出下一步改进的方向和目标。

Gstreamer框架是当前多媒体设计领域较成熟和常用的框架，基于该架构可以任意开发需要增添的媒体格式、传输协议的流媒体数据处理应用。基于Gstreamer的Chro­m­ium音视频播放系统不仅便于后期的扩展，而且被封装成共享库的插件所提供的元件可以被多个程序共享使用，降低开发成本。另外，此系统支持WebAPP的使用，对相关的第三方软件可以无缝对接。基于GStreamer 架构的Chromium音视频播放系统不仅能给用户提供更多、更好的音视频享受，还简化了集成和复用，快速响应市场需求，提高产品竞争力。

关键词 Chromium，Webkit，多媒体框架，Gstreamer

**THE DESIGN AND IMPLEMENTATION**

**OF CHROMIUM AUDIO AND VIDEO PLAYER**

**BASED ON GSTREAMER**

ABSTRACT

With the popularity of cars and the accelerated pace of life, people in their daily lives more and more use of cars. For automotive multimedia systems, the majority of consumers are no longer satisfied with the local audio and video playback. They want to enjoy Internet multimedia services anytime, anywhere in the vehicle terminal, such as openning the car terminal browser directly to browse web, playing favorite music and movie, or listenning to the latest music online. This subject analyzes the present situation and trend of embedded multimedia application, studies the open source framework and the related technology of the browser audio and video playing architecture, completes the research on the multimedia and multimedia terminal development platform, and lastly realizes Design and Implementation of Chromium Browser Based on Gstreamer for Audio and Video Playing System.

The main work of this paper includes:

1. On the base of having an understading about the current mainstream browsers, summarize its basic features. Systematically describes how the browser converts HTML, CSS, and JavaScript text document into a visual image interface, including the basic flow and working principle. Under these premises, take the rendering engine WebKit as an example to introduce its internal design, and make an overall analysis on Chromium that takes WebKit as the kernel.
2. Provides a detailed analysis of Media in Chromium, including its modules and realization. Give a detailed description and analysis about the related classes and sequences, and understand and master its internal implementation mechanism. Reconstruct and optimize the existing framework, according to the demand of the product.
3. In-depth study of Gstreamer framework and impaortant parts, detailly introduce core conceptes and basic programming skills, and briefly introduce its system integration methods, debugging methods, and call mechanism. Design and realize andio and video playing, and package the internel details, in order to provide it to the third party users and to develop secondary.
4. Connect Chromium and Gstreamer through inserting a middle layer, on the base of reconstructing Chromium Media and Gstreamer media playing.
5. Test and vertify design concept and implementation, show the existing problems, and put forward the direction and goal to improve next.

Gstreamer framework is a mature and common framework for multimedia design. It can be used to develop streaming media data processing applications those need to add media formats and transmission protocols. Based on the Gstreamer, Chromium audio and video playback system is easy to be extend later. Multiple programs can share plug-ins packaged as a shared library provided by the components, reducing development costs. In addition, this system supports the use of web application, and the relevant third-party software can be used without obstacles. GStreamer architecture-based Chromium audio and video playback system can not only provide users with more and better audio and video enjoyment, but also simplifies the integration and reuse, rapid response to market demand, and improve product competitiveness.

KeywordsChromium, WebKit, Multimedia framework, Gstreamer

**目 录**

[1 绪论 1](#_Toc467614449)

[1.1 研究背景及意义 1](#_Toc467614450)

[1.2 国内外研究现状 2](#_Toc467614451)

[1.3 本文工作 4](#_Toc467614452)

[1.4 论文组织结构 5](#_Toc467614453)

[2 相关技术 6](#_Toc467614454)

[2.1 浏览器 6](#_Toc467614455)

[2.1.1 浏览器特性 6](#_Toc467614456)

[2.1.2 浏览器基本工作原理 7](#_Toc467614457)

[2.1.3 WebKit内核及Chromium浏览器总体架构 9](#_Toc467614458)

[2.3 Gstreamer基础 13](#_Toc467614459)

[2.3.1 基本概念 13](#_Toc467614460)

[2.3.2 总线（Bus） 15](#_Toc467614461)

[2.3.3 Gstreamer使用方法 16](#_Toc467614462)

[2.4 本章小结 18](#_Toc467614463)

[3 需求分析 19](#_Toc467614464)

[3.1 功能需求 19](#_Toc467614465)

[3.1.1 一般性功能需求 19](#_Toc467614466)

[3.1.2 特殊性功能需求 26](#_Toc467614467)

[3.2 性能需求 29](#_Toc467614468)

[3.2.1 对于网络音视频文件 29](#_Toc467614469)

[3.2.2 对于本地音视频文件 30](#_Toc467614470)

[3.3 可靠性需求 30](#_Toc467614471)

[3.4 资源使用需求 30](#_Toc467614472)

[3.5 设计要求 30](#_Toc467614473)

[3.5.1 设计原则 30](#_Toc467614474)

[3.5.2 编码要求 30](#_Toc467614475)

[3.6 本章小结 30](#_Toc467614476)

[4 设计与实现 32](#_Toc467614477)

[4.1 Chromium中音视频框架分析 32](#_Toc467614478)

[4.2 设计 36](#_Toc467614479)

[4.2.1 概要设计 36](#_Toc467614480)

[4.2.2 Chromium Media Portability Implement分析与设计 37](#_Toc467614481)

[4.2.3 Media Service分析与设计 46](#_Toc467614482)

[4.2.4 AVChannel Manager分析与设计 48](#_Toc467614483)

[4.3实现 49](#_Toc467614484)

[4.3.1 Chromium Media Portability Implement相关的实现 49](#_Toc467614485)

[4.3.2 Media Service相关的实现 51](#_Toc467614486)

[4.3.3 AVChannel Manager相关的实现 53](#_Toc467614487)

[4.4 本章小结 53](#_Toc467614488)

[5 测试与优化 54](#_Toc467614489)

[5.1 功能测试 54](#_Toc467614490)

[5.1.1 测试用例与测试结果 54](#_Toc467614491)

[5.2 性能测试 58](#_Toc467614492)

[5.2.1 测试用例与测试结果 58](#_Toc467614493)

[5.3 优化 62](#_Toc467614494)

[5.3.1 占用资源的优化 62](#_Toc467614495)

[5.3.2 视频描画效率的优化 63](#_Toc467614496)

[5.4 本章小结 63](#_Toc467614497)

[6 总结与展望 64](#_Toc467614498)

[6.1 工作总结 64](#_Toc467614499)

[6.2 问题和展望 65](#_Toc467614500)

[参考文献 67](#_Toc467614501)

[致 谢 69](#_Toc467614502)

1 绪论

1.1 研究背景及意义

如今网络已经应用于人们日常生活中的各个环节，成为了其重要的组成部分。然而随着计算机网络通信技术、多媒体技术等的不断发展，静态的网页和简单的文字已经满足不了人们对网络信息的需求，人们更倾向于通过网络获取包含丰富多彩的音频和视频信息。为此，多媒体网络技术作为一种计算机综合处理动态图文信息的技术，凭借其很好的集成性、友好的交互性以及准确的实时性，将音频和视频的特性相结合[1]，将人们带到了一个丰富多彩的网络世界，如今这种技术已经得到广泛的应用，大大的便利了人们的日常生活。

当下，HTML5技术、音视频技术都已经被市场上主流浏览器支持，大多数的音视频网站也都同时提供Flash和HTML5两种页面。特别是Chrome浏览器对于HTML5的支持[2]，基于Chrome和HTML5的视频播放系统，可带来流畅的播控体验，并且浏览器性能消耗更低。这样的方法在嵌入式系统、移动设备中的互联网视频应用方案更能发挥其特有的优势。同时满足跨平台，高性能，体验友好的网站需求。

车载多媒体终端作为一款嵌入式移动设备， 被广泛搭在应用消费型轿车上[3]。随着移动互联的大力发展， 消费者已经不再满足于传统本地和外部存储介质（例如U盘）音视频文件播放，等等，更希望在车载移动终端上随时随地使用网络音视频播放服务。

在车载多媒体终端上搭载浏览器就显得很有必要，开源的Chromium浏览器（chr­om­e是Chromium的稳定版，不开源）以其精简的界面、快速的响应、流畅的体验和开源特性获得众多厂商的青睐。

然而， 鉴于多媒体系统功能复杂程度的提高，且不同的多媒体系统硬件平台和软件平台也存在着差异，这使得开发者必须为这些平台重新编写多媒体处理的代码。在开发应用程序时，开发人员也要花费大量的时问和精力来对这些底层的多媒体处理进行整合和使用[4]。

Gstreamer 框架是一种管道的设计思想[5]，基于该架构可以任意开发需要增添的媒体格式、传输协议的流媒体数据处理应用[6]。它提供的功能性元件可以有选择性的被安装到应用管道程序中是该架构最显著的优点，若采用GStreamer这个强大的通用多媒体框架，应用程序开发人员就可以将底层多媒体处理的整合和使用交给GStreamer，从而将精力集中于现有的媒体上，避免在如何处理后端、插件机制等问题上花费过多的时间。除此之外，被封装成的共享库的插件所提供的元件可以被多个应用程序共享使用[7]，省去了一些播放器代码的重复编写。基于GStreamer 架构的播放器不仅能给用户提供更多、更好的音视频享受，简化了集成和复用，快速响应市场需求，提高产品竞争力。

综上，在嵌入式车载多媒体设备上实现基于GStreamer框架的Chromium音视频播放在当今移动互联时代的工程应用上很有现实意义.

1.2 国内外研究现状

浏览器是指可以显示网页服务器或者本地文件系统的 HTML 文件内容，并让用户与这些文件交互的一种软件[8]。互联网的革命浪潮带动了众多技术的快速发展，网络浏览器作为互联网的最重要的终端接入口之一在短短的二十多年时间里日新月异。经过近二十年的发展，现在的浏览器已经有几十种之多。市场占有率在前面几位的主要有 IE、Firefox、Google Chrome、Safari、Opera 等[9]。特别是在进入21世纪后，越来越多的功能被加入到浏览器中来。在W3C等标准组织的积极推动下逐步形成HTML5技术，更成为了浏览器发展的火箭推进器。

当前，已经有很多浏览器成功扩展或移植到嵌入式平台的案例。较多的表现在互联网电视领域，例如基于Widget的Yahoo ConnectedTV，增加了 Web 页面的扩展功能，能够在Linux下的电视一体机终端或者机顶盒上运行，同时传统电视以及互联网功能也均可以正常应用。基于 Android 的 GoogleTV以 Android 系统和谷歌 Chrome 网络浏览器为基础，融合传统电视服务模式及丰富 Web 服务于一身的综合娱乐体验系统。GoogleTV 相比Yahoo ConnectedTV 的方案会略显复杂，除了涵盖软件系统外，还将外围设备与硬件整合在一起，以此丰富生态系统的完整性。

一般说来，浏览器都是基于浏览器排版引擎开发的，很少有人或公司从头开始编写浏览器。目前常用的浏览器引擎有如下几种[10,11]。

Microsoft Windows 操作系统的浏览器 IE（Internet Explorer）的排版引擎Trident。跨平台的、Mozilla Firefox浏览器的引擎Gecko，可以运行于 Microsoft Windows、Linux 和 Mac OS X 等操作系统之上。Konqueror浏览器所使用的KHTML，速度较快，缺点是对错误语法的容忍度低。开源浏览器引擎WebKit，由 KDE 的 KHTML 修改而来，现在使用它作为引擎的浏览器有 Google Chrome、Midori、Safari、Maxthon3 等，还有其他一些浏览器也计划转到 WebKit 引擎上来。Opera Software 开发的Presto，使用它的浏览器有 Opera、任天堂 DS、Wii 浏览器等。 Tasmam，微软推出的 IE for Mac 所使用的排版引擎。

2008年，Google公司以苹果开源项目WebKit作为内核，创建了一个新的项目Chromium，Chromium是谷歌开源网页浏览器计划的代号[12]。该项目的目标是创建一个快速的、支持众多操作系统的浏览器，包括对桌面系统和移动操作系统的支持。Chromium使用了同Safari一样的浏览器内核。在Chromium的基础上，Google发布了自己的浏览器产品Chrome。不同于WebKit之于Safari浏览器，Chromium本身就是一个浏览器，而不是Chrome浏览器的内核，Chrome浏览器一般会选择Chromium的稳定版本作为它的基础。Chromium是开源试验场，它会尝试很多创新的并且大胆的技术，当这些技术稳定之后，Chrome才会把它们集成进来。也就是说Chrome的版本会落后于Chromium；其次，Chrome还会加入一些私有的编码解码器以支持音视频等；再次，Chrome还会整合Google众多的网络服务[13]。自推出以，以其清爽精炼的用户界面、快速的渲染和脚本执行效率迅速壮大成为最流行的浏览器[14]。

目前，嵌入式系统面临一个巨大的发展机遇，后PC时代对嵌入式系统技术也提出了更广泛的需求[15]。嵌入式设备也凭着精湛的开发技术具备了许多原来所没有的功能，PC机上的许多应用也在嵌入式设备上得到了成功的移植和充分的应用[16]。在嵌入式系统技术快速发展的同时无线网络技术也日益成熟，以往单调的媒体服务也不再满足人们的生活需求。在线直播、网络视频等网络媒体服务随之发展起来，这也对嵌入式设备对多媒体信息处理的能力上也提出了更高的要求。

随着中国三大运营商不断推出各种移动网络业务，在一定程度促进了移动网络的大规模普及，使得移动多媒体业务有了更广阔的市场和前景[17]。在现阶段，作为一个移动智能设备，多媒体应用是一个必不可少的功能之一。多媒体的播放，包括本地文件和网络的流媒体，都需要强大的内置多媒体播放引擎的支持，这是多媒体功能满足用户体验的重要保证。

各种嵌入式设备上都有一些不同的浏览器和播放器等多媒体处理软件。现在媒体文件的格式种类非常多，每种格式的编解码都有不同的实现代码，从而使得嵌入式设备上的解码库也变得非常复杂[18]。并且这些解码库的安装也需要许多的技术支持，同时也会受到一些限制。因此，在嵌入式设备上开发播放器就会变得比较繁琐，如果针对每一种音视频文件都编写相应的解码程序就会变得非常困难也完全没有必要。

而Gstreame就为简化Linux下多媒体的开发提出了一种比较简洁的框架结构[19]。开发者可以利用这种框架结构设计一系列媒体处理模块来进行音频、视频处理。虽然Gstreamer的开发人员和开发资料比较少，具有一定的开发难度，但是相信这种多媒体框架以其简洁的开发机制和模块化的开发结构一定会成为Linux环境下主要的多媒体开发工具。

Gstreamer是流行于Linux系统下的多媒体应用程序开发框架，其采用了基于元件和管道的体系结构，框架中的所有的功能模块都被实现成可以插拔的组件，并且在需要的时候能够很方便地安装到任意一个管道上，由于所有插件都通过管道机制进行统一的数据交换，因此很容易利用各种插件组装出一个功能完善的多媒体应用程序[20]。

在嵌入式多媒体系统中，多媒体标准技术的各种实现方法，为嵌入式多媒体系统的发展起到了关键的作用。由于在嵌入式系统中，微处理器、存储器等资源的有限，在需要很高资源的多媒体标准技术的实现上遇到瓶颈。但随着 MPEG-1、MPEG-2、 MPEG-4、 H.261、 H.263 等主要的国际标准的硬件级实现，加快了嵌入式多媒体系统的发展。而目前中国的国家标准 AVS 标准，它能够提高编码效率[21]，这个也在一定程度上，解决了软硬件编解码上的效率。

1.3 本文工作

本文首先对当前市场占有率较高的常见浏览器进行了解和总结，概括了其基本特性；随后对具有代表性的浏览器的工作流程做了系统的阐述，并对工作原理做了进一步解析；然后以WebKit为对象对浏览器渲染引擎做了深入的模块解析，在此基础上对Chromium浏览器整体框架分析和了解。

其次，深入分析和研究Gstrea­me­r多媒体开发框架，对一些重要的组成部分，如元件、管道、箱柜、衬垫和总线做详细介绍，同时对Gstreamer的使用方法、运行机制做必要说明。另外对Chromium多媒体部分的设计与实现细节做了详细的了解和分析，针对核心部分和重要环节做出类图和序列图分析。在此基础上对Chromium的Media进行重构与改写，利用当前比较成熟的Gstrea­me­r体系对Chromium的多媒体部分进行新的设计与实现。包括需求分析、总体设计、详细设计、代码实现的部分细节。

最后，编写测试用例对本文所提到的设计方案和实现结果从功能和性能等角度进行了测试与验证，提出尚且存在的不足和下一步的工作方向。

1.4 论文组织结构

论文共分为六个章节，各章节的主要内容安排如下：

第1章，绪论部分。主要介绍了浏览器的发展、各浏览器引擎的基本概况、Chro­m­ium的历史背景、Gstreamer技术的基本思想，并阐述了该文的背景和意义以及相关技术研究成果和未来发展趋势。

第2章，基本理论和相关技术的介绍。介绍了浏览器的特性，并以主流的渲染引擎WebKit和开源的Chromium工程为重点分析了浏览器的工作原理，以及浏览器的架构的设计思想；比较全面的阐述了Gstreamer的基本概念、基本元件、框架理念、使用方法。。

第3章，需求分析。对车载车载多媒体终端要完成的功能、性能，和资源消耗等做了必要的需求分析。

第4章，设计与实现。基于第2章和第3章的系统介绍，对Chromium的源代码分析，进行了新的构建与布局，将Gstreamer应用于Chromium音视频播放系统。从源代码分析、到设计、到实现，完成整个工程的实现。

第5章，测试与优化。对第4章的成果进行测试与验证，并提出在嵌入式系统资源有限的情况下对系统进行优化的方案和实现。

第6章，总结与展望。总结了本文的研究成果与不足，并针对其功能扩展和应用领域提出了进一步的研究方向。

2 相关技术

2.1 浏览器

网页浏览器（英语：Web Browser，常被称为浏览器（Browser），之后将简称为浏览器）是一种用于检索并展示万维网信息资源的应用程序。这些信息资源可为网页、图片、影音、或其他内容，他们由统一资源标志符（Uniform Resource Identifier，缩写URI）标志。信息资源中的超链接可使用户方便地浏览相关信息。浏览器虽然主要用于使用万维网，但也可用于获取专用网络中网页服务器之信息或文件系统内之文件[22]。当前，微软IE、Mozilla火狐和Google Chrome成了桌面系统上最流行的三款浏览器，三者一起占据了该市场超过90%的浏览器份额。浏览器作为用户访问互联网最重要的接口，也难怪获得如此众多巨头的关注，未来，必将还是浏览器继续高速发展和竞争激烈的场景[23]。

2.1.1 浏览器特性

大体上来讲，浏览器的这些功能包括网络、资源管理、网页浏览、多网页管理、插件和扩展、书签管理、历史记录管理、设置管理、下载管理、账户和同步、安全机制、隐私管理、外观主题、开发者工具等。下面是对它们之中一些重要功能的简要介绍。

网络是第一步，浏览器通过网络模块来下载各种各样的资源，例如HTML文本，JavaScript代码、样式表、图片、音视频文件等。网络部分其实非常重要，因为它耗时比较长而且需要安全访问互联网上的资源。从网络上下载或者本地获取的资源，需要资源管理功能将他们管理起来，这需要高效的管理机制，例如如何避免重复下载资源、缓存资源等，都是它们需要解决的问题。网页浏览是浏览器的核心也是最基本、最重要的功能，它通过网络下载资源并从资源管理器获得资源，将它们转变为可视化的结果，这也是后面介绍的浏览器内核最重要的功能。很多浏览器支持多页面浏览，所以需要支持多个网页同时加载，这让浏览器变得更为复杂。同时，如何解决多页面的相互影响和安全等问题也非常重要，为此，一些浏览器做了大量的工作，例如可能使用线程或是进程来绘制网页。插件和扩展是现代浏览器的一个重要特征，他们不仅能显示网页，而且能支持各种形式的插件和扩展。插件是用来显示网页特定内容的，而扩展则是增加浏览器新功能的软件或压缩包。目前常见的插件有NPAPI插件、PPAPI插件、ActiveX插件等，扩展则跟浏览器由密切关系，常见的有Firefox扩展和Chromium扩展。账户和同步将浏览的相关信息，例如历史记录、书签等信息同步到服务器，给用户一个多系统下的统一体验，这对用户非常友好，是浏览器易用性的一个显著标识。安全机制本质是提供一个安全的浏览器环境，避免用户信息被各种非法工具窃取和破坏。这可能包括显示用户访问的网站是否安全、为网站设置安全级别、防止浏览器被恶意代码攻破等。开发者工具对普通用户来说用处不大，但对网页开发者来说意义却非比寻常。一个优秀的开发者工具可以帮助审查HTML元素、调试JavaSc­ript代码、改善网页性能等。

2.1.2 浏览器基本工作原理

以浏览器内核WebKit为例，根据数据的流向，可以将渲染分为三个阶段，第一个阶段是从网页的URL到构建完DOM树，第二个阶段是从DOM树到构建完成WebKit的绘图上下文，第三个阶段是从绘图上下文到生成最终的图像。为了描述这个过程，下面会将WebKit中的一些细节展示出来。图2-1描述的是从网页URL到构建完DOM树这个过程，数字表示的是基本顺序，当然也不是严格一致，因为整个过程可能重复并且可能交叉。

当用户在地址栏输入URL的时候，WebKit调用资源加载器加载该URL对应的网页，对应图中序号1；加载器依赖网络模块建立连接，发送请求并接收答复，对应序号2；WebKit接收到各种网页或者资源的数据，其中某些资源可能是同步或异步获取的，对应序号3；网页被交给HTML解释器转变成一系列的词语（Token），对应序号4；解释器根据词语构建节点（Node），形成DOM树，对应序号5；如果节点是Java­Script代码的话，调用JavaScript引擎解释并运行，对应序号6；解释器根据词语构建节点Node，形成DOM树，对应序号7；如果节点需要依赖其他资源，例如图片、CSS、视频等，调用资源加载器来加载它们，但是它们是异步的，不会阻碍当前DOM树的继续创建；如果是加载JavaScript资源URL（没有标记异步方式），则需要暂停当前DOM树的创建，直到JavaScript的资源加载并被JavaScript引擎执行后才继续DOM树的创建，对应序号8。

在上述的过程中，网页在加载和渲染过程中会发出“DOMContent”事件和DOM的“onload”事件，分别在DOM树构建完成后以及DOM树建完并且网页所依赖的资源都加载完之后发生，因为某些资源的加载并不会阻碍DOM树的构建，所以这两个事件多数时候不是同时发生的。接下来就是WebKit利用CSS和DOM树构建Render­Object树直到绘图上下文，如图2-2所示的过程。

资源加载器

URL

网页内容、JS、CSS等

DOM树

网络模块

HTML解释器

JavaScript引擎

1

2

3

4

5

6

7

8

图2-1 从网页URL到DOM树

Fig 2-1 the process of generating DOM tree

DOM树

CSS解释器

CSS

RenderObject树

RenderLayer树

绘图上下文

布局计算

1

2

3

图2-2 从CSS和DOM到绘图上下文

Fig 2-2 From CSS and DOM to Drawing context

这一阶段的具体过程如下：CSS文件被CSS解释成内部表示结构，对应序号1；CSS解释器工作完之后，在DOM树上附加解释后的样式信息，这就是RenderObject树，对应序号2；RenderObject节点在创建的同时，WebKit会根据网页的层次结构创建RenderLayer树，同时构建一个虚拟的绘图上下文，对应序号3。RenderObject树的建立并不表示DOM树会被销毁，事实上，上述图中的四个内部表示结构一直存在，直到网页被销毁，因为它们对于网页的渲染起了非常大的作用。

最后就是根据绘图上下文来生成最终的图像，这一过程主要依赖2D和3D图形库，如图2-3所示。图中这一阶段对应的具体过程如下：绘图上下文是一个与平台无关的抽象类，它将每个绘图操作桥接到不同的具体实现类，也就是绘图具体实现类，对应标号1；绘图实现类也可能有简单的实现，也有可能复杂的实现。在Chromium中，它的实现相当复杂，需要Chromium的合成器来完成复杂的多进程和GPU加速机制，对应标号2；绘图实现类将2D图形库或者3D图形库绘制的结果保存下来，交给浏览器来同浏览器界面一起显示，对应标号3。这一过程实际上可能不像图中描述的那么简单，现代浏览器为了绘图上的高效性和安全性，可能会在这一过程中引入复杂的机制。

绘图上下文

2D图形库

最终的图像

3D图形库

绘图具体实现类

3

1

2

图2-3 从绘图上下文到最终的图像

Fig2-3 From Drawing final image

2.1.3 WebKit内核及Chromium浏览器总体架构

WebKit的一个显著特征就是它支持不同的浏览器，因为不同浏览器的需求不同，所以在WebKit中，一些代码可以共享，但是另外一部分是不同的，这些不同的部分称为WebKit的移植（Ports）。图2-4是一个较为详尽的的WebKit架构图。

图中的WebKit架构，虚线框表示该部分模块在不同浏览器使用的WebKit内核中实现是不一样的，也就是它们不是普遍共享的。用实线框标记的模块表示它们基本上是共享的。之所以没有说的那么绝对，是因为它们中的一些特征性可能并不是共享的，而且可以通过不同的编译配置改变它们的行为。这里面有很多的不同，所以，很多使用WebKit的浏览器可能会表现出不同的行为。

图中最下面是“操作系统”，WebKit可以在不同的操作系统上工作。不同浏览器可能会依赖不同的操作系统，同一个浏览器使用的WebKit也可能依赖不同的操作系统，例如，Chromium浏览器支持Windows、MacOS、Linux、Android等系统。

在“操作系统”之上的就是WebKit赖以工作的第三方库，这些库是WebKit运行的基础。通常来讲，它们包括图形库、网络库、视频库等，加载和渲染网页需要它们不足为奇。WebKit是这些库的使用者，如何高效地使用它们是WebKit和各种浏览器厂商的一个重大课题，主要是如何设计良好的架构来利用它们以获得高性能。现代浏览器的功能越来越强大，性能要求也越来越高，新的技术不断被引入浏览器和Web平台，这也大大增加了WebKit和浏览器的复杂性。

WebKit

WebCore

WebKit嵌入式接口

WebKit2嵌入式接口

WebKit绑定

WebKit2绑定

WebKit Ports

网络栈

音视频

文字

硬件加速

图片解码

. . .

JavaScriptCore

CSS

SVG

布局

渲染树

HTML

DOM

Inspector

. . .

2D图形库

t

3D图形库

t

网络库

t

存储

t

音频库

t

视频库库

t

. . .

t

操作系统

图2-4 WebKit架构

Fig 2-4 the Architecture of WebKit

在它们二者之上的就是WebKit项目了，图中已经把它细分为两层，每层包含很多个模块，由于图的大小限制，略去了其中一些次要模块。

WebCore部分包含了目前被各个浏览器所使用的WebKit共享部分，这些都是加载和渲染的网页的基础部分，他们必不可少，具体包括HTML解释器、CSS解释器、SVG、DOM、渲染树（RenderObject树、RenderLayer树等），以及Inspector（Web Inspector、调试网页）。当然，这些共享部分有些是基础框架，其背后的支持也需要各个平台的不同实现。

JavaScriptCore引擎是WebKit中的默认JavaScript引擎，也就是说一些WebKit的移植使用该引擎。刚开始，她的性能不是很好，但是随着越来越多的优化被加入，现在的性能已变得非常不错。之所以说它还是默认的，是因为它不是唯一并且是可以替换的。事实上，WebKit中对JavaScript引擎的调用是独立于引擎的。在Google的Chromium开源项目中，它被替换为V8引擎。

WebKit Ports指的是WebKit中的非共享部分，对于不同浏览器使用的WebKit来说，移植中的这些模块由于平台差异、依赖的第三方库和需求不同等方面的原因，往往按照自己的方式来设计和实现，这就产生了移植部分，这是导致众多WebKit版本的行为并非一致的重要原因。这其中包括硬件加速架构、网络栈、视频解码、图片解码等。

在WebCore和WebKit Ports之上的层主要是提供嵌入式编程接口，这些嵌入式接口是提供给浏览器调用的（当然也可以有其他使用者）。图中有左右两个部分分别是狭义WebKit的接口和WebKit2（WebKit2 is a new API layer for WebKit to support a split process model, where the web content lives in a separate process from the application UI[24]）的接口。因为接口与具体的移植有关，所以有一个与浏览器相关的绑定层。绑定层上面就是WebKit项目对外暴露的接口层。实际上接口层的定义也是与移植密切相关的，而不是WebKit由是统一接口。

以上，便是对WebKit架构的介绍。

Chromium也是基于WebKit的，而且在WebKit的移植部分中，Chromium也做了很多有趣的事情，所以通过Chromium可以了解如何基于WebKit构建浏览器。另一方面，Chromium也是很多技术的创新者，它将很多先进的理念引入到浏览器领域。Chromium的代码非常复杂，模块非常多，结构也不是特别清晰，非常容易让人迷惑。为了方便理解，我们从架构和模块、多进程模型和多线程模型角度一一剖析。

首先要熟悉的是Chromium的架构及其包含的模块。图2-5描述了Chromium的架构和主要的模块。从图中可以看到，Blink只是其中的一块，和它并列的还有众多的Chromium模块，包括GPU/CommandBuffer（硬件加速架构）、V8 JavaScript引擎、沙箱模型、CC（Chromium Compositor）、IPC、UI等（还有很多并没有在图中显示出来）。在上面这些模块之上的就是著名的“Content模块”和“Content API（接口）”，它们是Chromium对渲染网页功能的抽象。“Content”的本意是指网页的内容，这里是指用来渲染网页的模块。这里或许有个疑问，WebKit不就是渲染网页内容的吗？是的，没有Content模块，浏览器开发者也可以在WebKit的Chromium移植上渲染网页内容，但是没有办法获得沙箱模型、跨进程的GPU硬件加速机制、众多的HTML5功能，因为这些功能很多是在Content层实现的。

“Content模块”和“Content API”将下面的渲染机制、安全机制、和插件机制等隐藏起来，提供一个接口层。该接口目前被上层模块或者其他项目使用，内部调用者包括Chromium浏览器、Content Shell等，外部包括CEF（Chromium Embedded Framework）、Opera浏览器等。

Chromium浏览器

Content Shell

Android WebView

Content接口

Content模块

Blink(WebKit)

GPU/Command Buffer

V8

沙箱模型

CC(Chromium合成器)

IPC、

UI、

PPAPI、

. . .

图2-5 Chromium模块结构图

Fig 2-5 Modules of Chromium

“Chromium浏览器”和“Content Shell”是构建在Content API之上的两个“浏览器”，Chromium具有浏览器完整的功能，也就是我们编译出来能看到的浏览器式样。而“Content Shell”是使用Content API来包装的一层简单的“壳”，但是它也是一个简单的“浏览器”，用户可以使用Content模块来渲染和显示网页内容。Content Shell的作用很明显，其一可以用来检测Content模块很多功能的正确性，例如渲染、硬件加速等；其二是一个参考，可以被很多外部的项目参考来开发基于“Content API”的浏览器或者各种类型项目。

以上，是我们对浏览器基本技术的简单介绍。当然，浏览器所涉及的技术不仅仅只有我们介绍的这些，它是一个庞大而复杂的项目。考虑到篇幅所限，我们只针对本论文可能涉及的部分进行了简单的描述，以便于后文的展开。

2.3 Gstreamer基础

Gstreamer是一个开源的多媒体框架库。利用它，可以构建一系列的媒体处理模块，包括从简单的Ogg/Vorbis回放、到音视频流媒体以及复杂的音频（混音）和视频（非线性编辑）处理。应用程序可以透明地利用解码和过滤技术，开发者可以使用简洁通用的接口来编写一个简单的插件来添加新的解码器或滤镜[25]。它的基本设计思想来自于俄勒冈（Oregon）研究生学院有关视频管道的创意，并且借鉴了微软DirectSho­w的设计思想[26]。Gstreamer框架是基于插件的，有些插件中提供了各种各样的多媒体数字信号编解码器，也有些提供了其他的功能。所有的插件都能够被链接到任意的已经定义了的数据流管道中。它的核心库函数是一个处理插件、数据流和媒体操作的框架。另外，核心库还提供了一个API，这个API是开放给开发者使用的，当开发者需要使用其他的插件来编写他所需要的应用程序的时候可以使用它。Gstreamer的程序开发框架使得编写任意类型的流媒体应用程序成了可能。在编写处理音频、视频或者两者皆有的应用程序时，Gstreamer可以让工作变得简单。它并不受限于音频和视频处理，理论上能够处理任意类型的数据流。管道设计的方法对于实际应用的滤波器几乎没有负荷，它甚至可以用来设计出对延时有很高要求的高端音频应用程序。

2.3.1 基本概念

在Gstreamer中有一些基本的概念和术语，了解和理解这些概念和术语是使用Gstreamer的前提和基础，后续章节会进场提到这些概念和术语。

最重要的一个概念就是元件（elements）。要想使数据流顺利的传输，需要创建一系列的元件，并将这些元件按照一定规则连接起来（当然连接元件涉及到例外一个概念，后边会介绍。），然后数据流就可以在这些被连接起来的元件间传输。想象一下一段段的自来水管，为了把洁净的水输送到每家每户，工作人员从供水中心铺设管道，利用大小不一各种各样的接口、水管和其他元件一段衔接一段接入到每个家庭。元件的类型有多种，一个元件它的类型可能是sources、filters、muxers、demux­ers、codecs或者sinks。每个元件都有一个特殊的函数接口，有些元件的函数接口是用于能够读取文件的数据、解码文件数据的，而有些元件的函数接口只是输出相应的数据到具体的设备上（例如：声卡设备、显卡设备，等）。我们将若干个元件连接在一起，从而创建一个管道来完成一个特殊的任务，例如音频的播放和录制。Gstreamer默认已经安装了很多有用的元件，通过使用这些元件你能够构建一个具有多种功能的应用程序。在某些情况下，开发人员可以根据项目的需要编写一些特别的新的元件。一般来说，元件像是一个黑盒，数据从黑盒的一端流入，在黑盒内部进行一些加工处理，然后从另外一端流出被处理过的数据。例如：对于一个mux­er元件，流入的是视频文件、音频文件、可能还会有字幕文件，流出的是某一格式的视频文件。如，可将1.avi、1.mp3、1.srt用muxer合并为mkv格式的视频文件。

插件（plugins）是一块可以加载的代码。通常被称为共享对象文件（shared oject file）或动态链接库（dynamically linked library）[27]。一个插件中可以包含一个或多个元件，因此可以把多个元件封装进一个插件中，使之支持新的功能。Gstreamer充斥着插件的概念，即使你只用到一些标准的包。核心库只有少量的基本函数，其它所有的功能都由插件来实现。一个XML文件被用来保存所有注册的插件的详细信息。这样，使用Gstreamer的程序可以只在需要时加载插件，而不必实现全部加载。

衬垫（pads）是Gstreamer的另外一个基本概念。在Gstreamer中，衬垫是用来在元件间协商连接和数据流的。衬垫可以看做元件见相互连接的“接口”，数据流通过这些接口流入和流出元件[28]。衬垫具有特殊的数据处理能力：衬垫可以限制通过它的数据类型。只有当两个衬垫允许通过的数据类型兼容时才可以将它们连接起来。打个比方，衬垫类似于物理设备上的a plug or jack。想想一个包含功放、DVD播放器和一个视频投影仪的家庭影院系统。将投影仪和DVD播放器相连是允许的，因为这两个设备具有兼容的video jack。而将投影仪和功放连接起来也许就行不通了，因为他们之间的jack不同。大部分情况下，所有在Gstreamer中流经的数据都遵循一个原则，数据从元件的一个或多个源衬垫流出，从一个或多个sink衬垫流入。源和sink元件分别只有源和sink衬垫。

以上是基本概念和术语的介绍，下面我们介绍几种常见的元件：

1. 源元件（source elements）

源元件是管道数据的产生者，比如从硬盘或者声卡等设备中将数据读取出来。源元件有且只有一个源衬垫（source），它只产生数据，并不接收数据。也就是说，它是数据的源头，不能对数据做任何处理。

1. 过滤器（filters）与类过滤元件（Filter-like elements）

过滤器以及类过滤元件都同时拥有输入和输出衬垫，它们对从输入衬垫接收到的数据进行处理操作，然后将处理后的数据提供给输出衬垫。音量元件、视频转换器、Ogg分流者或者Vorbis解码器都是这种类型的元件。类过滤元件可以拥有任意个的元衬垫或者接收衬垫。像一个视频分流器可能有一个接收衬垫以及多个（1-N）源衬垫，每个接收衬垫对应一种元数据流（elementary stream）。相反地，解码器只有一个源衬垫及一个接收衬垫。

1. 接收元件（sink elements）

接收元件处在媒体管道的末端，它只有一个接收衬垫（sink pad），因而它只接收数据，不产生数据。例如常见的写磁盘、利用声卡播放声音，以及视频输出都是由接收元件实现的。

1. 箱柜（bins）和管道（pipelines）

箱柜是一个可以装载元件（element）的容器。管道是箱柜的一个特殊子类型，管道可以操作包含在它自身内部的所有元件。因为箱柜本身又是元件的子集，所以能够像操作普通元件一样地操作箱柜，通过这种方法可以降低应用程序的复杂度。开发者可以改变一个箱柜的状态来改变箱柜内部所有元件的状态。箱柜可以发送总线消息（bus messages）给它的子集元件。管道是高级的箱柜，当设定管道的暂停或者播放状态的时候，数据流将开始流动，并且媒体数据处理也开始进行。一旦开始，管道将在一个单独的线程运行，直到被停止或者数据流播放完毕。

2.3.2 总线（Bus）

总线是一个简单的系统，它拥有一套自己的线程机制，采用这套机制可以把一个管道线程的message分发到一个应用程序中。当使用Gstreamer的时候，由于总线机制的存在，应用程序不需要对线程过分关心或者线程识别，尽管Gstreamer可能已经加载了多个线程。

当一个管道创建后，默认是包含有一个总线的，因此应用程序不需要再创建总线，只需要在总线上设置一个消息处理器。当主循环运行的时候，总线会轮询这个消息处理器看是否有新的消息，如果有消息则消息被采集到，总线会调用相应的回调函数完成处理。

使用总线的方法有两种[29]。第一种是运行GLib/Gtk+主循环（也可以自己运行默认的GLib循环），然后使用侦听器对总线进行侦听。如果使用这种方法，Glib的主循环将轮询总线上是否存在新的消息，当存在新的消息的时候，总线会马上通知你。在这种情况下，会用到gst\_bus\_add\_watch()/gst\_bus\_add\_signal\_watch()两个函数。当使用总线时，设置消息处理器到管道的总线上可以使用gst\_bus\_add\_watch()来创建一个消息处理器来侦听管道。每当管道发出一个消息到总线，这个消息处理器就会被触发，消息处理器则开始检测消息信号类型从而决定那些时间将被处理。第二种是自己侦听总线消息，使用gst\_bus\_peek()/gst\_bus\_poll()就可以实现。

Gstreamer有几种由总线传递的预定义消息类型，这些消息都是可扩展的，插件可以定义自己的一些信息[30]。所有的消息都有一个消息源、类型和时间戳。这个消息源能被用来判断由哪个element发出消息。例如在众多的消息中，应用程序只对上层的管道发出的消息感兴趣（例如状态发生了改变）。

2.3.3 Gstreamer使用方法

下面，我们以一个具体的实例来看看如何利用Gstreamer框架的组件，实现一个简单的MP3播放器。数据源元件负责从磁盘上读取数据，过滤器元件负责对数据进行解码，而接收器元件负责将解码后的数据写入声卡。

要想在程序中应用Gstreamer提供的各种功能，首先必须在主函数中调用gst\_init()来完成初始化工作，以便用户从命令行输入的参数传递给Gstreamer函数库。一个典型的Gstreamer应用程序的初始化如下所示[31]：

#include <gst/gst.h>

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

{

gst\_init(&argc, &argv);

}

接下去需要创建三个元件并连接成管道，由于所有Gstreamer都具有相同的基类GstElement，所以可以采用如下方式进行定义：

GstElement \*pipeline, \*filesrc, \*decoder, \*audiosink;

管道在Gstreamer框架中是用来容纳和管理元件的，下面的代码将创建一条名为pipeline的新管道：

pipeline = gst\_pipeline\_new(“pipeline”);

数据源元件负责从磁盘文件中读取数据，它具有名为“location”的属性，用来指明文件在磁盘上的位置。使用标准的GObject属性机制可以为元件设置相应的属性：

filesrc = gst\_element\_factory\_make(“filesrc”, “disk\_source”);

g\_object\_set(G\_OBJECT(filesrc), “location”, argv[1], NULL);

过滤器元件负责将解码后的数据利用声卡播放出来：

audiosink = gst\_element\_factory\_make(“audiosink”, “play\_audio”);

已经创建好的三个元件需要全部添加到管道中，并按顺序连接起来：

gst\_bin\_add\_many(GST\_BIN(pipeline), filesrc, decoder, audiosink, NULL);

gst\_element\_link\_many(filesrc, decoder, audiosink, NULL);

所有准备工作都做好之后，就可以通过将管道的状态切换到PLAYING状态，来启动整个管道的数据处理流程：

gst\_element\_set\_state(pipeline, GST\_STATE\_PLAYING);

 这里可以通过gst\_pipeline\_get\_bus(GST\_PIPELINE(pipeline))获取到总线bus，然后加入一个消息处理函数bus\_call来监视产生的消息：

gst\_bus\_add\_watch(bus, bus\_call, loop)

bus\_call可以定义如下：

static gboolean bus\_call(GstBus \*bus, GstMessage \*msg, gpointer data)

{

// do something

}

 当播放结束，终止管道，并解除引用释放资源：

gst\_element\_set\_state(pipeline, GST\_STATE\_NULL);

gst\_object\_unref(GST\_OBJECT(pipeline));

以上是一个简单的使用Gstreamer创建播放器的实例，便于我们理解Gstreamer的使用方法和流程。

2.4 本章小结

本章首先对浏览器的历史沿革做了简单的介绍，以便了解开发浏览器的初衷与目的；其次以现代浏览器为主，描述浏览器具备的特性；在对浏览器充分了解的基础上，随后展开对浏览器工作原理的进行阐述；然后以WebKit和Chromium为重点，详细分析了浏览器的核心架构与设计。此外，还对Gstreamer由浅入深，由简单到复杂地展现了其基本概念、使用方法、工作机制、等等，以便对Gstreamer有全面的认识。以上，为下文的展开，做了系统性的铺垫。

3 需求分析

3.1 功能需求

3.1.1 一般性功能需求

1 管理控制

基于Gstreamer的Chromium音视频播放系统应具备基本的播放管理控制功能，即常见的播放、暂停、停止、跳转、监测、调节音量、错误处理等，相关用例图如图3-1所示。



图3-1 管理控制用例图

Fig3-1 Management & control use case diagram

在图3-1中，最左侧表示Chromium浏览器使用者，包括自然人和其他被Chrom­ium浏览器支持的应用等；中间的表示Chromium的音视频播放子系统，它对右边的Media Service和AV Channel Manager存在一定的依赖关系。关于图3-1中涉及到的用例的具体的分析如下。

|  |
| --- |
| 用例名称：播放 |
| 描述：此用例主要描述浏览器用户打开带有音频或视频的页面，按下播放按钮或者网页音视频被设置为自动播放时，浏览器加载音视频文件进行播放。 |
| 用例编号：UC1 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：网络已连接，浏览器已启动。 |
| 主事件流：   1. 用户打开带有<audio>或<video>标签的网页，网页加载完毕   A1：页面中的audio或video被设置成自动播放   1. 用户点击播放按钮 2. 页面开始加载音视频文件   A2：网速较慢   1. 当数据加载满足播放条件，会看到播放开始 2. 用例结束 |
| 其他事件流：  A1：页面中的audio或video标签被设置成自动播放   1. 无需用户点击播放按钮，自动进入资源加载状态 2. 进入主事件留第3步   A2：网速较慢   1. 出现旋转的网络加载进行中按钮 2. 返回主事件流第3步 |
| 后置条件：进入到播放状态，进度条开始更新 |
| 备注：无 |

|  |
| --- |
| 用例名称：暂停 |
| 描述：此用例主要描述当此系统已经进入到正常播放状态后，点击网页上的暂停按钮，此系统会进入暂停状态中。 |
| 用例编号：UC2 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：用于已经成功播放多媒体文件，处于播放状态。 |
| 主事件流：   1. 当前音频或视频文件已进入播放状态，且正常播放中 2. 点击控制条上的暂停按钮 3. 系统进入到暂停状态 4. 用例结束 |
| 其他事件流：无 |
| 后置条件：状态由播放进入到暂停，进度条停止更新 |
| 备注：无 |

|  |
| --- |
| 用例名称：停止 |
| 描述：此用例用于描述当已经进入到播放或暂停状态中，点击停止按钮，此系统会释放缓存的资源，并进入到停止状态。 |
| 用例编号：UC3 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：用于已经成功播放多媒体文件，处于播放或暂停状态。 |
| 主事件流：   1. 当前音频或视频文件已进入到播放状态或暂停状态 2. 点击控制条上的停止按钮 3. 系统进入到停止状态 4. 用例结束 |
| 其他事件流：无 |
| 后置条件：状态由播放或暂停，进入到停止，时间进度条归零。 |
| 备注：此种状态下缓存的资源被完全释放，如果再次点击播放按钮，需重新从网上load资源。 |

|  |
| --- |
| 用例名称：静音 |
| 描述：此用例主要描述网页内包含音频或视频标签时，点击控制条上的静音按钮，音量会立即调整为0 |
| 用例编号：UC4 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：网络已连接，检测到页面内有音视频标签 |
| 主事件流：   1. 打开的网页内含有音频或视频资源，点击控制条上的静音按钮 2. 静音图标显示，音量被降到最低   A1：点击播放按钮进行播放音频或视频   1. 再次点击静音按钮，音量恢复正常 2. 用例结束 |
| 其他事件流：  A1：点击播放按钮进行播放音频或视频   1. 如果是音频资源则只看到进度更新却听不到声音，如果是视频资源则只看到画面和进度条更新却听不到声音。 2. 返回主事件流第3步 |
| 后置条件：无 |
| 备注：无 |

|  |
| --- |
| 用例名称：调节音量 |
| 描述：此用例主要描述在音视频播放过程中，滑动音量调节条，会听到音量的大小变化。 |
| 用例编号：UC5 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：网络已连接，检测到页面内有音视频标签 |
| 主事件流：   1. 打开的网页内含有音频或视频资源，滑动音量调节条 2. 会看到音量数值大小的变化，如果有声音正在播放，会听到音量的变化 3. 用例结束 |
| 其他事件流：无 |
| 后置条件：音量随调节条滑动变化 |
| 备注：无 |

|  |
| --- |
| 用例名称：跳转 |
| 描述：此用例主要描述在视频在播放或暂停过程中，点击进度条上的进度滑块并拖动到任意时间位置，松开后将从该位置开始播放。 |
| 用例编号：UC6 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：音视频资源在播放或暂停状态中，网络连接良好。 |
| 主事件流：   1. 音视频资源正在播放中，滑动时间更新进度条上的滑块 2. 在进度条上的任意位置，松开进度滑块 3. 播放系统将自动从松开的时间位置开始播放 4. 用例结束 |
| 其他事件流： 无 |
| 后置条件：系统从跳转的时间点开始播放 |
| 备注：无 |

|  |
| --- |
| 用例名称：获取播放进度 |
| 描述：此用例用来描述对于处在播放状态中的音视频文件资源，可以获取当前的播放进度。 |
| 用例编号：UC7 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：音视频资源正在播放过程中 |
| 主事件流：   1. 播放音频或视频文件，播放状态持续中 2. 可以随着播放的进行，进度条在正常的更新位置和时间 3. 用例结束 |
| 其他事件流：无 |
| 后置条件：无 |
| 备注：无 |

|  |
| --- |
| 用例名称：获取总时长 |
| 描述：此用例用于描述当浏览器检测到有音视频文件资源的时候，会在资源的基本信息中读取总时长。 |
| 用例编号：UC8 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：网络连接状态良好、Chromium浏览器正常运行。 |
| 主事件流：   1. 打开含有<audio>或<video>标签的网页 2. 点击播放按钮，待浏览器获取文件资源 3. 在控制条右侧自动显示文件总时长 4. 用例结束 |
| 其他事件流：无 |
| 后置条件：音视频资源总时长一直显示 |
| 备注：无 |

|  |
| --- |
| 用例名称：获取播放状态 |
| 描述：此用例用于描述各种状态的变化，包括：播放、暂停、停止、加载中、错误 |
| 用例编号：UC9 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：Chromium浏览器正常启动 |
| 主事件流：   1. 打开包含有<audio>或<video>标签的页面   A1：网络无连接  A2：网速较慢   1. 点击播放按钮进入到播放状态 2. 点击暂停按钮进入到暂停状态 3. 点击停止按钮进入到停止状态 4. 用例结束 |
| 其他事件流：  A1：网络无连接   1. 弹出网络无连接的提示窗口 2. 返回主事件流第5步   A2：网速较慢   1. 弹出加载进度旋钮 2. 显示加载进度 3. 返回到主事件流第5步 |
| 后置条件：状态显示 |
| 备注：无 |

|  |
| --- |
| 用例名称：获取文件信息 |
| 描述：此用例用于描述在播放某些音视频资源的时候，可以获取到与该资源相关的信息，比如：标题、歌手、专辑名称、比特率，等等。 |
| 用例编号：UC10 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：音视频资源中包含相关的文件信息 |
| 主事件流：   1. 打开包含有文件信息的音视频资源进行播放 2. 可以在播放控制条下方看到标题、歌手、专辑名称等文件信息 3. 用例结束 |
| 其他事件流：无 |
| 后置条件：文件信息暂时在控制条下方 |
| 备注：无 |

|  |
| --- |
| 用例名称：事件通知 |
| 描述：此用例主要用于描述在音视频播放过程中，若是有高优先级的事件（比如电话打入）发生的时候，会暂停当前音视频的播放，响应高优先级的事件。 |
| 用例编号：UC11 |
| 参与者：用户、Chromium、通道资源管理模块 |
| 前置条件：音视频资源文件正在播放中 |
| 主事件流：   1. 音频或视频正在播放中 2. 收到高优先级事件 3. 暂停当前音视频资源的播放 4. 用例结束 |
| 其他事件流：无 |
| 后置条件：播放中的音视频资源被暂停 |
| 备注：无 |

|  |
| --- |
| 用例名称：错误处理 |
| 描述：此用例主要用于描述当播放的音视频资源出现无法解码或者其他未预知到的错误时，需要立即反馈给用户。 |
| 用例编号：UC12 |
| 参与者：用户、Chromium |
| 前置条件：Chromium正常开启 |
| 主事件流：   1. 打开浏览器，播放音视频文件 2. 出现不能解码或是其他类型错误 3. 弹出警示框告知用户错误类型 4. 用例结束 |
| 其他事件流：无 |
| 后置条件：弹出警告窗口 |
| 备注：无 |

2 格式支持

音视频播放系统的数据处理是系统能否正常工作的前提，本系统对音视频文件格式及编码格式的支持需求如下表（表3-1）所示。

表3-1 支持文件格式与编码格式

Table 3-1 the supported file format and encoding format

|  |  |  |
| --- | --- | --- |
| 类型 | 支持文件格式 | 支持编码方式 |
| 音频 | MP3，AAC，OGG | AAC，MP3，Vorbis |
| 视频 | MP4，AVI，MKV，WEBM，3GP，TS，OGG | H.264，MPEG-4 |

相关用例分析如下。

|  |
| --- |
| 用例名称：音频格式支持 |
| 描述：此用例用于描述系统对音频文件格式和编码格式的支持。 |
| 用例编号：UC13 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：浏览器正常运行 |
| 主事件流：   1. 用户打开网络或本地音频资源 2. 分别播放文件格式MP3、AAC、OGG，编码方式AAC、MP3、Vorbis的音频文件 3. 以上音频文件均能正常播放 4. 用例结束 |
| 其他事件流：无 |
| 后置条件：以上文件格式和编码方式的音频资源都正常播放 |
| 备注：无 |

|  |
| --- |
| 用例名称：视频格式支持 |
| 描述：此用例用于描述系统对视频文件格式和编码格式的支持。 |
| 用例编号：UC14 |
| 参与者：用户、Chromium浏览器 |
| 前置条件：浏览器正常运行 |
| 主事件流：   1. 用户打开网络或本地视频资源 2. 分别播放文件格式MP4、AVI、MKV、WEBM、3GP、TS、OGG，编码方式H.264、MPEG-4视频文件 3. 以上视频文件均能正常播放 4. 用例结束 |
| 其他事件流：无 |
| 后置条件：以上文件格式和编码方式的视频资源都正常播放 |
| 备注：无 |

3.1.2 特殊性功能需求

1 Web App的支持

系统除了要支持Chromium浏览器的本地及网络音视频的播放外，还需支持Web APP相关音视频的播放。

用例分析如下。

|  |
| --- |
| 用例名称：Web APP的支持 |
| 描述：Web APP是基于Web的应用，它依赖于本地浏览器的支持。本用例主要用于描述基于Gstreamer的Chromium音视频播放系统对多媒体类型的Web APP的支持。 |
| 用例编号：UC15 |
| 参与者：用户、Chromium浏览器、WebAPP |
| 前置条件：网络连接良好、系统正常运行 |
| 主事件流：   1. 用户打开已近安装好的Web APP 2. 点击应用中的音频或视频文件进行播放 3. 音视频文件正常播放 4. 用例结束 |
| 其他事件流：无 |
| 后置条件：Web APP音视频文件正常播放 |
| 备注：无 |

2 传统播放器的支持

系统除了要支持Chromium浏览器的本地及网络音视频的播放及Web APP相关音视频的播放，还需要对本地传统播放器支持，比如本地独立播放器只用于播放插入的USB或者SD卡内的音视频资源文件。其功能包括：获取文件列表、上一首或下一首跳转、快进或快退、倍速播放，其用例图如图3-2所示。



图3-2 对传统播放器的支持用例图

Fig3-2 Supporting traditional player use case diagram

用例分析如下。

|  |
| --- |
| 用例名称：获取文件列表 |
| 描述：本用例主要用于描述在给定的路径下，扫描音视频文件的功能。 |
| 用例编号：UC16 |
| 参与者：用户、传统本地播放器、本系统 |
| 前置条件：本系统正常运行 |
| 主事件流：   1. 打开本地播放器，输入指定的路径 2. 系统自动扫描该路径下的音视频文件 3. 将扫描结果返回给本地播放器 4. 用例结束 |
| 其他事件流：无 |
| 后置条件：播放时显示音视频文件列表 |
| 备注：无 |

|  |
| --- |
| 用例名称：上/下一首 |
| 描述：本用例主要用于描述在已经获取是音视频资源列表的情况下，选择上一首或下以后进行播放。 |
| 用例编号：UC17 |
| 参与者：用户、传统本地播放器、本系统 |
| 前置条件：本系统正常运行 |
| 主事件流：   1. 本地播放器已近获取播放列表，且正在播放中 2. 点击本地播放器的上一首或下一首按钮   A1：无法播放   1. 播放列表里的上一个或下一个音视频文件开始播放 2. 用例结束 |
| 其他事件流：  A1：无法播放   1. 显示无法播放的警告3秒 2. 自动跳转到该文件的下一个自动播放 3. 返回主事件流的第3步 |
| 后置条件：以上文件格式和编码方式的视频资源都正常播放 |
| 备注：无 |

|  |
| --- |
| 用例名称：快进/快退 |
| 描述：本用例主要用于描述在正常播放状态下，点击快进/快退时的情景。 |
| 用例编号：UC18 |
| 参与者：用户、传统本地播放器、本系统 |
| 前置条件：本系统正常运行 |
| 主事件流：   1. 传统本地播放器正常播放中 2. 用户点击快进会快退按钮 3. 系统以进度跳转的形式快速前进或后退 4. 用例结束 |
| 其他事件流：无 |
| 后置条件：进入快进/快退模式 |
| 备注：无 |

|  |
| --- |
| 用例名称：倍速播放 |
| 描述：本用例主要用于描述在正常播放状态下，点击倍速按钮，进入倍速播放的状态 |
| 用例编号：UC19 |
| 参与者：用户、传统本地播放器、本系统 |
| 前置条件：本系统正常运行 |
| 主事件流：   1. 传统本地播放器正常播放中 2. 用户点击倍速播放一次 3. 系统进入加速播放状态，速度是原来的2倍 4. 再次点击倍速播放，播放倍速加到原来的4倍 5. 再次点击倍速播放，返回到正常播放 6. 用例结束 |
| 其他事件流：无 |
| 后置条件： 倍速播放 |
| 备注：无 |

3.2 性能需求

3.2.1 对于网络音视频文件

（1）当发起网络加载时，加载响应时间不超过1秒。

（2）当网络信号较弱或网速较慢时，依据情况进行缓冲，并每隔10毫秒更新一次缓冲进度。

（3）若发生网络中断或遇到解码失败的情况，立即通知Error。

（4）若要求暂停时，软件响应时间最长不超过1秒。

（5）要求停止后，10秒后释放缓存所占资源空间。

3.2.2 对于本地音视频文件

（1）当要求对音视频文件资源进行播放时，软件响应时间最长不能超过2秒。

（2）当要求暂停时，软件响应时间最长不能超过1秒。

（3）播放过程中，要求解码迅速、播放不卡顿、音质良好、画面流畅。

3.3 可靠性需求

（1）连续运行一周无故障概率99.5%。

（2）如果发生Crash，能够立即复归。

3.4 资源使用需求

运行期间，占用最大内存不高于60M。

3.5 设计要求

3.5.1 设计原则

在将软件需求转换成设计模型的过程中，不仅要考虑如何实现功能需求，而且要考虑和关注软件的可扩充性、可维护性、可重复性和通用性，采用抽象、分解和模块化、封装和信息隐藏、高内聚和低耦合的原则进行设计。

3.5.2 编码要求

遵循开发流程，在设计的指导下进行代码编写，程序要具有良好的结构，提高程序的封装性，减低程序的耦合程度。代码要具有良好的可读性，易于理解，方便调试和测试；易于使用和维护，良好的修改性、扩充性、可重用性强、移植性好；在不降低可读性的前提下，尽量提高代码的执行效率和减少资源占用。

3.6 本章小结

本章对需求做了言简意赅的精要分析，提出了在功能、性能、可靠性、资源占用等方面需满足的标准和交付要求，对第4章的设计与实现提出了具体的约束，具有指导性意义。

4 设计与实现

4.1 Chromium中音视频框架分析

WebKit提供了支持多媒体规范的基础框架，如音视频元素、JavaScript接口和视频播放等[32]。图4-1所示是WebKit支持视频的基础类和关系。



图4-1 WebKit支持视频的基础类和关系

Fig 4-1 The base classes supported by WebKit and their relationship

图中左侧HTMLMediaElement和HTMLVideoElement是DOM树中的节点类，分别对应W3C标准中的定义，包含众多接口，这些接口可以被JavaScript代码访问。

其次是MediaPlayer和MediaPlayerClient两个类，他们的作用非常明显。MediaPlayer是一个公共标准类，被HTMLMediaElement类使用来播放音频和视频，它本身只是提供抽象接口，具体实现依赖于不同的WebKit移植。同时，一些播放器的状态信息需要通知到HTMLMediaElement类，这里使用MediaPlayerClient类来定义这些有关状态信息的接口，HTMLMediaElement类需要继承MediaPlayerClient类并接收这些状态消息。根据前面的描述，规范要求将事件派发到JavaScript代码中，而这一实现在HTMLMediaEle­ment类完成。

然后是是不同移植对MediaPlayer类的实现，其中包括MediaPlayerPrivateInterface类和WebMediaPlayerClientImpl类。前者是除了Chromium移植之外使用的标准接口，也是一个抽象接口，由不同的移植来实现。后者是Chromium移植的实现类。为什么会这样？因为Chromium将WebKit复制出Blink最后就将MediaPlayerPrivateInterface类直接移除了，而在MediaPlayer类中直接调用它。WebMediaPlayerClientImpl类会使用Chromium移植自己定义的WebMediaPlayer接口来作为实际的播放器，而真正的播放器则是在Chromium项目的代码中来实现[33]。

最后是同渲染有关的部分，这里面包含RenderObject树和RenderLayer树，图中RenderMedia类和RenderVideo是RenderObject的子类，用于表示Media节点和Video节点。图4-1是采用硬件加速机制的视频播放所使用的类，WebKit也支持使用软件渲染方式来播放视频，当需要绘制的时候，由RenderVideo类使用HTMLMediaElement类获取MediaPlayer对象，调用它的paint方法来让MediaPlayer对象将解码后的图像绘制在2D图像上下文接口中。

因为视频资源相对其他资源而言，一般比较大，当用户播放视频的时候，需要连续性播放以获得较好的体验，但是网络可能不是一直都稳定和高速，所以资源的获取对用户体验很重要，需要使用缓存机制或者其他机制来获取视频资源[34]。

图4-2是Chromium中缓存资源类。BufferDataSource类表示资源数据，它是一个简单的数据表示类，内部包含一个较小的内存空间，实际的缓冲机制由BufferResourceLoader类来完成。



图4-2 Chromium的视频资源缓存类

Fig 4-2 The cache classes supported by WebKit and their relationship

根据多进程架构的设计原则，Chromium的媒体播放器的实现应该在Renderer进程，而对于资源的获取，应该在Browser进程[35]。Chromium支持媒体播放器的具体实现相当复杂，而且涉及到不同的操作系统，目前Chromium在不同的操作系统上实现的媒体播放器也不一样。图4-3是Chromium基础类，为了方便理解这些类和图4-1中类之间的关系，图4-3标注了一些WebKit中同Chromium直接相关的类。



图4-3 Chromium支持视频的基础类和关系

Fig 4-3 The base classes supported by WebKit and their relationship

图4-3上半部分是WebKit和WebKit的Chromium移植中的相关类，下半部分是Chromium中使用硬件加速机制来实现视频播放的基础设施类。而从左往右看，左边部分是播放器的具体实现类，右边部分则是支持视频在合成工作中的相关类。

首先来看一看这些类和对象的创建过程。WebMediaPlayerClientImpl类是WebKit在创建HTMLMediaElement对象之后创建MediaPlayer对象的时候，由MediaPlayer对象来创建的。当视频资源开始加载的时候，WebKit创建一个WebMediaPlayer对象，当然就是Chromium中的具体实现类WebMediaPlayerImpl对象，同时WebMediaPlayerClientImpl类也实现了WebMediaPlayerClient类，所以WebMediaPlayerImpl在播放视频的过程中需要向该WebMediaPlayerClient类更新各种状态，这些状态信息最终会传递到HTMLMediaElement类中，最终可能成为JavaScript事件[36]。之后，WebMediaPlayerImpl对象会创建一个WebLayerImpl对象，还会同时创建VideoLayer对象，根据合成器的设计，Chromium还有一个LayerImpl树，在同步的时候VideoLayer对象对应的VideoLayerImpl对象会被创建。之后Chromium需要创建VideoFrameProviderClientImpl对象，该对象将合成器的Video层同视频播放器联系起来并将合成器绘制一帧的请求转给提供视频内容的VideoFrameProvider类，这实际上是调用Chromium的媒体播放器WebMediaPlayerImpl，因为它就是一个VideoFrameProvider类的实现子类。

然后是Chromium如何使用这些类来生成和显示每一帧的。当合成器调用每一层来绘制下一帧的时候，VideoFrameProviderClientImpl::AcquireLockAndCurrentFrame()函数会被调用，然后该函数调用WebMediaPlayerImpl类的GetCurrentFrame函数返回当前一帧的数据。VideoLayerImpl类根据需要会将这一帧数据上传到GPU的纹理对象中。当绘制完这一帧之后，VideoLayerImpl调用VideoFrameProviderClientImpl::PutCurrentFrame来通知播放器这一帧已绘制完成，并释放掉相应的资源。同时，媒体播放器也可以通知播放器这一帧已绘制完成，并释放掉相应的资源。同时，媒体播放器也可以通知合成器有一些新帧生成，需要绘制出来，它首先调用播放器的VideoFrameProvider::DidReceiveFrame()函数，该函数用来检查当前有没有VideoLayerImpl对象，如果有对象存在，需要设置它的SetNeedsRedraw标记位，这样，合成器就知道需要重新生成新的一帧。

最后是上述有关视频播放对象的销毁过程。有多种情况使Chromium需要销毁媒体播放器和相关资源，如video对象被移除或者设置为隐藏等，这样视频元素对应的各种层对象，以及WebKit和Chromium中的这些设施就会被销毁。

在桌面系统中，Chromium使用了一套多媒体播放框架，而不是直接使用系统或者第三方库的解决方案[33]。图4-3是Chromium在桌面系统中采用的多媒体播放引擎的工作模块和过程，这一框架称为多媒体管线化引擎，图中主要的模块是多路分配器(Demuxer)、音视频解码器、音视频渲染器。这些部分主要是被WebMediaPlayerImpl类调用。



图4-4 Chromium的多媒体管线化引擎

Fig 4-4 The multimedia pipeline engine of Chromium

在处理音视频的管线化过程中，需要解码器和渲染器来分别处理视频和音频数据。它们均采用一种叫做“拉”而不是“推”的方式进行，也就是说由视频或者音频渲染器根据声卡或者时钟控制器，按需要来请求解码器解码数据，然后解码器和渲染器又向前请求“拉”数据，直到请求从视频资源文件读入数据。根据之前的多进程架构和Chromium的安全机制，整个管线化引擎虽然在Renderer进程中，但是由于Renderer进程不能访问声卡，所以图中渲染器需要通过IPC将数据或者消息同Browser进程通信，由Browser进程来访问声卡。

4.2 设计

4.2.1 概要设计

根据需求分析，在整体上可以分为三个部分：

Chromium Media Portability Implement

Media Service

AVChannel Manager

硬件

图4-5 整体模块划分

Fig 4-5 The overall modules

Chromium Media Portability Implement：它是与Chromium关联最强的部分，通过他联通了Chr­­­­­om­iu­­­m和Media Service的桥梁。当Chromium需要创建一个播放器来播放音视频文件的时候，此处的播放器被创建，与Media Service通信。

Media Service是随系统启动的一个进程，等待使用者的连接和播放控制，是真正的音视频播放的执行者。

AVChannel Manager是多媒体通道资源管理者及调配者，比如管理着声音类型的优先级，各种中断，等等。

4.2.2 Chromium Media Portability Implement分析与设计

当Chromium渲染引擎遇到<video> tag时，会创建对应的Element对象HTML­VideoElement；同样地，当遇到<audio> tag的时候，会创建HTMLAudioElement对象。HTMLVideoElement和HTMLAudioElement都是从HTMLMediaElement派生，他们之间的类关系如图4-6所示。



图4-6 Chromium中媒体类派生关系

Fig 4-6 Media derived relationships in Chromium

从上图（图4-6）可以看出，在HTMLMediaElement是HTMLAudioElement和HTMLVideoElement的父类，中拥有一个Media播放器，当音视频需要播放的时候，会通过MediaPlayer进行。从上图中我们可以看到，WebMediaPlayerClient­Impl是Media­Player的子类，负责Media播放的实现。而其实，WebMediaPlayerClientImpl也不是Media播放的真正实现者，WebMediaPlayerClientImpl又把播放的任务委托给了WebMedia­Player，如下图（图4-7）所示，WebMediaPlayer其实也只是一个接口类，根据不同的移植有不同的实现，图4-7中WebMediaPlayer右侧的WebMediaPlaye­rClient是播放状态的监听者，由图4-6中的WebMediaPlayerClientImpl继承实现，从而把播放状态层层向上反馈，最终在浏览器的页面上体现出来，比如播放进度的更新，等等。

对于图4-7中的各个类做如下说明：

WebMediaPlayer：接口类，被WebMediaPlayerSmartAuto实现，提供给Chromium使用。

WebMediaPlayerClient：接口类，由Chromium实现，主要用于简体player的状态及通知事件，并做出相应动作。

图4-7 Chromium Media的实现

Fig 4-7 the implementation of Chromium Media



WebMediaPlayerSmartAuto：WebMediaPlayer的实现者，实现WebMediaPlayer的所有接口。

RenderFrameImpl：是Chromium源代码中的类，我们在这个类里创建了Web­M­ediaPlayerSmartAuto。

RendererMediaPlayerManager：对WebMediaPlayerSmartAuto进行管理的管类

IPC：进程间通信相关，Chromium源生代码。

MediaWebContentsObserver：接收和过滤Render进程发出的Media消息。

BrowserMediaPlayerManager：对MediaPlayerSmartAuto进行管理。

MediaPlayerSmartAuto：Browser进程中的Media播放器。

WebMedia：接口类，被WebMediaImpl实现。

WebMediaImpl：实现WebMedia，中间件，负责Chromium和本地libMedia（NMCl­assicMediaPlayer）的衔接。

NMClassicMediaPlayer：可以看做Media Service提供的proxy类，给WebMediaImp使用。

PlayerListener：监听NMClassicMediaPlayer的状态变化及事件通知。

WebMediaCb：接口类，被WebMediaCbImpl实现。

WebMediaCbImpl：实现WebMediaCb，向MediaPlayerSmartAuto通知各种事件及状态变化。

AVChannelClient：资源管理的监听者，当监听到声音通道被抢占、视频交替、后者高优先级Source切换的变化后通知给Player。

NCSubChannelManager：对资源管理的子集类

AVChannelManager：管理各Channel，声音及视频通道的打开、关闭、切换、状态维护等。

在上图中，WebMediaSmartAuto是针对Chromium的WebMediaPlayer的实现者。在RenderFrameImpl的函数createMediaPlayer()中，完成了对播放引擎（在Chromium中有一套完善的播放引擎创建的规则，这里的引擎可以理解为播放器）的创建。关于WebMediaSmartAuto的I/F接口一览，我们可以参看以下表4-1。

表4-1 WebMediaSmartAuto I/F一览

Table4-1 WebMediaSmartAuto interface list

|  |  |  |  |
| --- | --- | --- | --- |
| **FunctionName** | **Param** | **Return** | **备注** |
| load | WebURL | void | 加载文件 |
| play | void | void | 播放 |
| pause | void | void | 暂停 |
| paint | Rect | void | 视频描画 |
| paused | void | bool | 判断是否暂停 |
| currentTime | void | double | 获取当前播放进度 |
| duration | void | double | 获取音视频文件总时长 |
| networkState | void | NetworkState | 获取网络状态 |
| readyState | void | ReadyState | 获取当前准备状态 |
| OnTimeUpdate | double | void | 进度更新通知 |
| OnDurationChanged | double | void | 总时长更新通知 |
| UpdateReadyState | ReadyState | void | 准备状态变更通知 |
| UpdateNetworkState | NetworkState | void | 网络状态变化通知 |
| playbackStateChanged | void | void | 播放状态变更通知 |

以上各函数的详细说明如下所示

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | load | |
| 処理概要 | | 加载文件 | |
| 参数 | 1 | WebURL | 网络音视频资源文件的路径 |
| 返値 | | void |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | play | |
| 処理概要 | | 播放 | |
| 参数 | 0 | void |  |
| 返値 | | void |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | pause | |
| 処理概要 | | 暂停 | |
| 参数 | 0 | void |  |
| 返値 | | void |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | paint | |
| 処理概要 | | 描画视频 | |
| 参数 | 1 | Rect | 描画区域 |
| 返値 | | void |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | paused | |
| 処理概要 | | 判断是否暂停 | |
| 参数 | 0 | void |  |
| 返値 | | bool | true:暂停 false:非暂停 |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | currentTime | |
| 処理概要 | | 获取当前播放进度 | |
| 参数 | 0 | void |  |
| 返値 | | double | 当前播放时间,单位sec |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | duration | |
| 処理概要 | | 获取音视频文件总时长 | |
| 参数 | 0 | void |  |
| 返値 | | double | 音视频文件总时长, 单位sec |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | OnTimeUpdate | |
| 処理概要 | | 进度更新通知 | |
| 参数 | 1 | double | 当前进度 |
| 返値 | | void |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | OnDurationChanged | |
| 処理概要 | | 总时长更新通知 | |
| 参数 | 1 | double | 音视频文件总时长 |
| 返値 | | void |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | UpdateReadyState | |
| 処理概要 | | 准备状态变更通知 | |
| 参数 | 1 | ReadyState | 准备状态定义  ReadyStateHaveNothing：  没有关于音频/视频是否就绪的信息  ReadyStateHaveMetadata：  关于音频/视频就绪的元数据  ReadyStateHaveCurrentData：  关于当前播放位置的数据是可用的，但没有足够的数据来播放下一帧  ReadyStateHaveFutureData：  当前及至少下一帧的数据是可用的  ReadyStateHaveEnoughData：  可用数据足以开始播放 |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | readyState | |
| 処理概要 | | 获取准备状态 | |
| 参数 | 0 | void |  |
| 返値 | | ReadyState | 准备状态的定义参看函数UpdateReadyState中ReadyState的定义 |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | UpdateNetworkState | |
| 処理概要 | | 网络态变更通知 | |
| 参数 | 1 | NetworkState | 网络状态  NetworkStateEmpty：  音频/视频尚未初始化  NetworkStateIdle：  音频/视频是活动的且已选取资源，但并未使用网络  NetworkStateLoading:  浏览器正在下载数据  NetworkStateLoaded：  加载完毕  NetworkStateFormatError：  格式错误  NetworkStateNetworkError：  网络错误  NetworkStateDecodeError：  解码错误 |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | networkState | |
| 処理概要 | | 获取网络状态 | |
| 参数 | 0 | void |  |
| 返値 | | NetworkState | 网络状态的定义参看函数UpdateNetworkState中NetworkState的定义 |

|  |  |  |  |
| --- | --- | --- | --- |
| 関数名称 | | playbackStateChanged | |
| 処理概要 | | 播放状态变更通知 | |
| 参数 | 0 | void |  |
| 返値 | | void |  |

从播放引擎被创建到开始播放的Sequence如图4-8所示，首先在播放器WebMe­dia­PlayerSmartAuto在Render­Frame­Impl被创建，在WebMe­dia­PlayerSmartAuto中load函数被调用的时候，会通过动态加载函数创建WebMediaImpl，然后对WebMediaImpl进行初始化操作，在初始化函数initPlayer中创建NMClassicMediaPlayer，并调用其加载函数。然后，WebMediaImpl调用AVChannelManager的active函数激活通道，开始play。当播放器进入paly状态后，把PLAYING状态通过回调函数playStateChanged()通知给Chromium浏览器，这时候用户在界面上看到的音视频文件在播放。

在音视频被播放的过程中，如果有电话接入、或者有交通警报等高优先级声音切入，就会发生声音通道被抢占的情况，此时，需要暂停Media的播放，其Sequence图如图4-9所示。首先，NCSubChannelManager将声音通道被抢占的通知发送给WebMediaPlayer；然后WebMediaPlayer调用pause函数暂停播放，并把暂停状态通知给WebMedia­Player­­SmartAuto，WebMediaPlayer­SmartAuto将pause状态通知给Chromium浏览器，用户看到播放器被暂停，高优先级的声音切入。

图4-8 播放器从被创建到开始播放的序列图

Fig4-8 the sequence of Media player being created and start



图4-9 声音通道被抢占的序列图

Fig4-9 the sequence diagram of the sound channel is occupied

4.2.3 Media Service分析与设计

我们希望所设计出的音视频播放系统除了支持Chromium音视频的播放外，还能支持本地等其他播放应用的需求。所以，我们把Media播放的业务实现做成一个Service运行于独立的进程，当客户端需要播放音视频的时候，就是用进程间通讯的方式和Serv­ice进行交互，图4-10展示了Media Service的类图结构。

在4.2.2小节中，我们提到WebMediaImpl使用了一个类叫NMClassic­MediaPlayer，这个类可以看做Media Service对外的client，当客户端需要使用播放的时候，可以使用此类和Service进行交互，进程间的通讯对于使用者来说是透明的。

在NMClassic­MediaPlayer中封装Service的proxy，即NMMediaPlayerProxy，它负责通过进程间通讯和Media Service的业务实现进行交互。进程间通讯的实现在不同的平台有不同的实现，例如在Android系统中可以使用binder完成。

NMMediaPlayerService是进程间通讯的另外一端，接收和处理NMMediaPlayerProxy发来的各种请求，并把结果再通过NMClassicMediaPlayerListener通知给调用者。

NMMediaPlayerClientManager是Service端播放器的管理者，真正播放器的创建，是由NMMediaPlayerClientInst完成。NMUnifiedPlayer会根据不同的需求创建NMStream­PlayerImpl或NMGstPlayerImpl。

在NMStream­PlayerImpl和NMGstPlayerImpl中使用Gstreamer插件完成对音视频流媒体和文件的解码播放。类图的最下方是外部依赖的库，由硬件供应商提供。



图4-10 Media Service 类图

Fig4-10 the class diagram of media service

4.2.4 AVChannel Manager分析与设计

AVChannel Manager模块控制与裁决source优先级的管理模块，根据不同的情况调整哪个source播放声音，哪个source暂停播放，这里的source指的是音频源或视频源。图4-11展现的是AVChannel Manager模块的类关系。



图4-11 AVChannel Manager 类图

Fig4-11 the class diagram of AVChannel Manager

在上图（图4-11）中，AVChannelManager是提供给用户使用的类，被Media Service模块的WebMediaImpl调用，假设WebMediaImpl要开始播放一段视频资源，首先要使用AVChannelManager的函数void activate(MediaTypeMask types, OutputMask outputs)来激活声音通道，当申请的声音通道被激活后，会在notifyStatusChangedAsync(const std::string &audioName, const std::string &status)通知状态变化，然后就可以开始播放了。

在AVChannelManager中封装了NCSubChannelManager，AVChannelManager将调度的实现委托给是NCSubChannelManager，NCSubChannelManager是进行source调度的实现者。

NCSubChannelListener是NCSubChannelManager的回调类，将状态的变化以及对NCSubChannelManager函数调用的结果反馈给调用者，调用者在使用NCSubChannel­Manager前需要使用实现NCSubChannelListener。

AVChannelClient是NCSubChannelListener子类，实现对NCSubChannelManager的通知回调处理，在AVChannelManager被创建的时候，AVChannelManager会创建一个AVChannelClient，然后把它注册到NCSubChannelManager。

NCSubChannelPluginCtrl是更底层的类，和硬件通讯，控制音视频通道的开启与关闭。它依赖于第三方库libnmavmplugin，为便于使用，是对依赖库的二次封装。

4.3实现

上一节介绍了基于Gstreamer的Chromium音视频播放系统的分析与设计，本节主要论述其实现过程。关于Chromium源码的获取及移植不是本文重点，这里不再展开，具体可参考Chromium Project官方网站<https://www.chromium.org>，有比较详细的说明。

4.3.1 Chromium Media Portability Implement相关的实现

在图4-7中，我们看到在类RenderFrameImpl中有一个函数是createMediaPlayer，这里便是Chromium的Media被创建的地方。注释掉原来创建函数的代码，改为我们自己的创建播放器代码，此函数中我们调用自己创建player的函数CreateSmartAutoWeb­Media­Player() ，在此函数中，创建WebMediaPlayerSmartAuto对象，并把它交给Renderer­MediaPlayerManager对象进行管理。WebMediaPlayer­SmartAuto是WebMediaPlayer的子类，函数返回值是WebMediaPlayer的指针。

当WebMediaPlayerSmartAuto对象被创建完成，它的load函数将会被调用。在WebMediaPlayerSmartAuto的load函数中，完成初始化操作，此过程是一个跨进程的操作。首先，我们上面看到的RenderFrameImpl对象和WebMediaPlayerSmartAuto对象，它们均运行于Render进程，这里的WebMediaPlayerSmartAuto本子上来讲只是一个proxy，它没有真正的业务逻辑的实现，所有动作都是抛到另外一个进程，即Browser进程完成。

在load被调用的时候，内部函数InitializePlayer被调用，在InitializePlayer通过Renderer­MediaPlayerManager发送一个初始化的消息：

Send(new MediaPlayerHostMsg\_Initialize(routing\_id(), media\_player\_params))

此消息在另外一个独立进程Browser中被MediaWebContentsObserver接收，Media­WebContentsObserver接到初始化消息后调用BrowserMediaPlayerManager的OnInitialize函数。

在BrowserMediaPlayerManager的OnInitialize函数中，创建MediaPlayerSmartAuto对象，然后调用MediaPlayerSmartAuto的load函数。MediaPlayerSmartAuto被创建时，在其构造函数中，通过动态函数加载动态库libWebMedia.so：

dlopen("libWebMedia.so", RTLD\_NOW)

利用这个动态库，创建WebMediaImpl对象：

m\_createWebMedia = (createObjFunc \*) dlsym(m\_dlHandle, "createObj");

createObj的实现在WebMediaImpl.cpp文件里实现：

extern "C" nutshell::web::WebMedia \*createObj(nutshell::web::WebMediaCb \*cb)

{

return new nutshell::web::WebMediaImpl(cb);

}

至此，便完成了从Render进程到Browser进程Media Player的创建过程。通过上述介绍可以看出，Render进程的Player只是一个影子，它通过Chromium独特的进程间通信的方式，在Render进程中将一切的函数请求转换成消息抛给Browser进程，Browser进程收到消息后对消息进行处理，在Browser进程里创建了另外一个Meidia Player来响应Render进程的对应请求。

之所以这样做，一是因为Chromium的多进程架构设计，关于Chromium的多进程架构可以参看前面第二章中的描述，尤其在图2-6中进行了形象化的说明。二是因为一个Chromium实例可能会有多个Render进程，但只会有一个Browser进程，将Media Play的业务实现放在Browser进程便于管理。

4.3.2 Media Service相关的实现

Media Service是一个独立的服务，在系统启动时候被创建，然后等待客户端来连接。在MediaServiceMain的实现中，首先创建NMMediaServerProc进程：

NMMediaServerProc process(argc, argv)

然后对其进行初始化处理：

process.initialize(NMMediaServiceFac)

开启进程：

process.start()

进入到进程loop中，等待连接通信：

process.enterloop()

如果进程遇到终止的条件，则跳出loop，调用stop函数：

process.stop()

最后是反初始化：

process.deinitialize()

Media Service运行起来后，就处于等待处理的循环中。

在上一节提到，MediaPlayerSmartAuto会创建WebMediaImpl对象，当WebMedia­Impl被创建，就会使用NMClassicMediaPlayer中的NMMediaPlayerProxy与NMMedia­Player­Service通信。NMMediaPlayerService通过NMMediaPlayerClientManager创建NMUnifiedPlayer，在NMUnifiedPlayer构造函数中根据feature的不同，来创建不同的Player进行处理。

如果检测到是流媒体的feature，就创建NMStreamPlayerImpl进行播放。在NMStream­PlayerImpl的prepareImpl函数中创建gloop和上下文：

m\_glibLoop->initLoop()

创建管道类：

createPipelineClass(&m\_pipelineClass, m\_mediaPath.c\_str(), NULL,

"pulsesink", "iautovideosink", NULL, elementCreateCallBack, this)

连接bus总线：

attachBusCall(GST\_ELEMENT\_CAST(m\_pipelineClass->pipeline()), m\_glibLoop->getGlibContext(), (GSourceFunc)busCall, this)

开启loop：

m\_glibLoop->startLoop()

如果检测到不是流媒体的feature，就创建NMGstPlayerImpl进行播放。在NMGst­PlayerImpl的prepareImpl函数中首先调用gstPrepare函数，gstPrepare完成gloop和上下文的创建：

m\_glibLoop->initLoop()

创建管道类：

createPipelineClass(&m\_pipelineClass, m\_mediaPath.c\_str(), NULL,

"pulsesink", "iautovideosink", NULL, elementCreateCallBack, this)

获取管道引用：

m\_pipeline = GST\_ELEMENT\_CAST(m\_pipelineClass->pipeline())

连接bus总线：

m\_busCallSource->attachBusCall(m\_pipeline, m\_glibLoop->getGlibContext(), (GSourceFunc)busCall, this)

开启loop：

m\_glibLoop->startLoop()

当调用到gstPlay()的时候，首先查看当前的播放状态，如果是PLAYING状态就直接返回，否则就调用setGstState改变状态开始播放：

setGstState(GST\_STATE\_PLAYING, NC\_TRUE, STATE\_CHANGE\_TIMEOUT, NC\_TRUE)

setGstState函数则是对gst\_element\_set\_state(m\_pipeline, state)的封装。

4.3.3 AVChannel Manager相关的实现

AVChannelManager在WebMediaImpl函数调用，用于通道的申请。当AVChannel­Manager的init函数被调用的时候，会通过调用NCSubChannelManager的registerAvm函数完成source的注册，即：

mNCSubChannelManager->registerAvm(NCString(WEBMEDIA\_AUDIO\_NAME), mClient)

当需要激活通道的时候，调用函数：

mNCSubChannelManager ->reqAudioSource(NCString(WEBMEDIA\_AUDIO\_NAME), true)

在NCSubChannelManager内部，会调用NCSubChannelPluginCtrl来实现，其本质是通过uComLib与硬件进行交互，uComLib是硬件供应商提供的动态库，此处不再展开。

4.4 本章小结

本章提出了基于Gstreamer的Chromium音视频播放系统的总体架构，将总体架构进行模块划分，并提出个模块的详细设计方案，对于比较重要的部分用类图及序列图做了必要的说明，在实现中对重要的代码和逻辑进行了代码级讲解，并阐述了某些关键函数的实现方法，对整个开发过程做了提纲挈领的描述。

5 测试与优化

5.1 功能测试

5.1.1 测试用例与测试结果

|  |
| --- |
| 测试用例名称：网络音频播放 |
| 测试用例描述：此用例用来测试在网络连接状态良好的状态下，使用Chromium播放网络音频文件的效果。 |
| 测试用例编号：TC1 |
| 前提条件：网络连接状况良好，浏览器已启动。 |
| 测试流程：   1. 打开常见的音频网站，例如虾米音乐、百度音乐 2. 进入到歌曲列表画面，点击其中的一首音乐进行播放 |
| 特殊流程：无 |
| 期望：能听到音乐声音播出，进度条根据播放进度进行更新。 |
| 测试结果：OK |
| 备注：测试截图如下 |

|  |
| --- |
| 测试用例名称：本地音频播放 |
| 测试用例描述：此用例用来测试在使用Chromium打开本地音频文件播放的效果。 |
| 测试用例编号：TC2 |
| 前提条件：浏览器已启动，准备好测试用的音频文件test.mp3 |
| 测试流程：   1. 拖动本地音频文件test.mp3拖到Chromium内 2. 等待播放 |
| 特殊流程：无 |
| 期望：能听到音乐声音播出，进度条根据播放进度进行更新。 |
| 测试结果：OK |
| 备注：测试截图如下 |

|  |
| --- |
| 测试用例名称：网络视频播放 |
| 测试用例描述：此用例用来测试在网络连接状态良好的状态下，使用Chromium播放网络视频文件的效果。 |
| 测试用例编号：TC3 |
| 前提条件：网络连接状况良好，浏览器已启动。 |
| 测试流程：   1. 打开常见的视频网站，例如youtube、搜狐视频 2. 进入到视频列表画面，点击其中的一个视频进行播放 |
| 特殊流程：无 |
| 期望：能看到视频画面正常播放，声音正常播出，进度条根据播放进度进行更新。 |
| 测试结果：OK |
| 备注：测试截图如下 |

|  |
| --- |
| 测试用例名称：本地视频播放 |
| 测试用例描述：此用例用来测试在使用Chromium打开本地视频文件播放的效果。 |
| 测试用例编号：TC4 |
| 前提条件：浏览器已启动，准备好测试用的视频文件movie.ogg |
| 测试流程：   1. 拖动本地视频文件movie.ogg到Chromium内 2. 等待播放 |
| 特殊流程：无 |
| 期望：能看到movie的画面流畅播放，声音正常播出，进度条根据播放进度进行更新。 |
| 测试结果：OK |
| 备注：测试截图如下 |

|  |
| --- |
| 测试用例名称：管理控制 |
| 测试用例描述：此用例用来测试在Chromium播放音视频过程中，点击暂停、调节音量、静音按钮的效果。 |
| 测试用例编号：TC5 |
| 前提条件：浏览器已启动，在正常播放音频或视频 |
| 测试流程：   1. 用浏览器打开音频或视频文件，进入到正常播放状态 2. 点击暂停按钮，观察播放状态的改变 3. 如果已经进入到播放状态，再次点击播放按钮，重新回到播放状态 4. 播放过程中，调节音量，听声音的变化 5. 点击静音按钮，听声音的变化 6. 再次点击静音按钮，听声音是否恢复 |
| 特殊流程：无 |
| 期望：各控制状态良好，响应及时 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：跳转 |
| 测试用例描述：此用例用来测试在Chromium播放音视频过程中，拖动进度条按钮到任意时间播放的效果。 |
| 测试用例编号：TC6 |
| 前提条件：浏览器已启动，在正常播放音频或视频 |
| 测试流程：   1. 用浏览器打开音频或视频文件，进入到正常播放状态 2. 拖动进度条的时间更新按钮到任意时间点 3. 观察是否从该时间点开始播放 |
| 特殊流程：无 |
| 期望：从跳转到的时间点开始播放 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：事件通知 |
| 测试用例描述：此用例用来测试在Chromium播放音视频过程中，当有优先级高的事件发生的时候的Notification。 |
| 测试用例编号：TC7 |
| 前提条件：浏览器已启动，在正常播放音频或视频 |
| 测试流程：   1. 用浏览器打开音频或视频文件，进入到正常播放状态 2. 模拟电话接入 3. 观察有播放状态有无转到暂停状态 |
| 特殊流程：无 |
| 期望：当有高优先级时间发生的时候，播放状态变为暂停。 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：错误处理 |
| 测试用例描述：此用例用来测试在Chromium播放音视频过程中，当有错误发生时的处理情况。 |
| 测试用例编号：TC8 |
| 前提条件：浏览器已启动，在正常播放音频或视频 |
| 测试流程：   1. 断开网络连接，观察有无warning弹出 2. 打开中间有损坏的文件，观察有无warning弹出 3. 打开不支持的音视频文件，观察有无warnig弹出 |
| 特殊流程：无 |
| 期望：当发生错误的时候，有相应的warning弹出。 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：Web App播放测试 |
| 测试用例描述：此用例用来测试Web APP在此系统播放音视频的情况。 |
| 测试用例编号：TC9 |
| 前提条件：网络连接良好，Web APP包含音视频文件。 |
| 测试流程：   1. 打开相应的Web APP 2. 点击其中的音频进行播放 3. 点击其中的视频进行播放 |
| 特殊流程：无 |
| 期望：Web APP内音视频正常播放 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：传统播放器的支持 |
| 测试用例描述：此用例用来测试本地播放器在此系统播放音视频的情况。 |
| 测试用例编号：TC10 |
| 前提条件：已有基于本系统开发的本地播放器 |
| 测试流程：   1. 插入存放有音视频文件的USB 2. 打开播放器 3. 获取文件列表 4. 选择其中一个进行播放 5. 点击上一首/下一首，选择播放 6. 点击快进/快退，观察进度 7. 点击倍速播放，观察播放速度 |
| 特殊流程：无 |
| 期望：对本地播放此支持良好，响应迅速。 |
| 测试结果：OK |
| 备注：无 |

5.2 性能测试

5.2.1 测试用例与测试结果

|  |
| --- |
| 测试用例名称：网络加载响应时间 |
| 测试用例描述：此用例用来测试Chromium在播放网络音视频的时候，加载响应时间 |
| 测试用例编号：TC11 |
| 前提条件：网速较慢，Chromium正常开启 |
| 测试流程：   1. 打开含有音视频资源的网页 2. 单击播放按钮，观察加载响应时间是否在1秒之内 |
| 特殊流程：无 |
| 期望：响应时间在1秒内 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：网络加载进度更新频率 |
| 测试用例描述：此用例用来测试Chromium在加载网络音视频的时候，网速较慢的情况下，进度更新频率 |
| 测试用例编号：TC12 |
| 前提条件：网速较慢，Chromium正常开启 |
| 测试流程：   1. 打开含有音视频资源的网页 2. 单击播放按钮，看到加载旋钮转动，同时显示加载进度 3. 用log查看，是否每10ms打印一次log，更新一次加载进度的通知 |
| 特殊流程：无 |
| 期望：每10ms更新一次加载频率 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：网络连接中断Error通知 |
| 测试用例描述：此用例用来测试Chromium在播放网络音视频的过程中遇到网络中断Error时的通知 |
| 测试用例编号：TC13 |
| 前提条件： Chromium正常开启，网络连接正常 |
| 测试流程：   1. 打开含有音视频资源的网页 2. 点击播放按钮播放音视频文件 3. 断开网络，观察有无网络连接中断Error画面弹出 |
| 特殊流程：无 |
| 期望：有网络连接中断Error画面弹出 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：解码失败Error通知 |
| 测试用例描述：此用例用来测试Chromium在播放网络音视频的过程中遇到解码失败Error时的通知 |
| 测试用例编号：TC14 |
| 前提条件： Chromium正常开启，网络连接正常 |
| 测试流程：   1. 打开含有音视频资源（超出解码能力）的网页 2. 点击播放按钮播放音视频文件 3. 观察有无解码失败Error画面弹出 |
| 特殊流程：无 |
| 期望：有解码失败Error画面弹出 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：暂停网络音视频的响应时间 |
| 测试用例描述：此用例用来测试Chromium在播放网络音视频的过程中要求暂停时的响应时间 |
| 测试用例编号：TC15 |
| 前提条件： Chromium正常开启，网络连接正常 |
| 测试流程：   1. 打开含有音视频资源的网页 2. 点击播放按钮播放音视频文件 3. 点击暂停按钮，观察是否在1秒内暂停播放 |
| 特殊流程：无 |
| 期望：1秒内暂停播放 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：停止网络音视频播放后资源释放 |
| 测试用例描述：此用例用来测试Chromium在播放网络音视频的过程中要求停止释放后，是否在10秒后释放缓存资源 |
| 测试用例编号：TC16 |
| 前提条件： Chromium正常开启，网络连接正常 |
| 测试流程：   1. 写一个包含音视频停止按钮的测试网页放在测试服务器上 2. 点击播放按钮播放音视频文件 3. 点击停止按钮，观察10秒后是否释放所占缓存资源 |
| 特殊流程：无 |
| 期望：停止10秒后释放所占缓存资源 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：本地音视频资源文件播放响应时间 |
| 测试用例描述：此用例用来测试播放本地音视频资源时候的响应时间 |
| 测试用例编号：TC17 |
| 前提条件：准备好本地音视频资源文件 |
| 测试流程：   1. 使用Chromium浏览器或者基于此系统开发的本地播放器打开准备好的音视频文件资源 2. 观察软件响应时间 |
| 特殊流程：无 |
| 期望：2秒内进入播放状态 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：本地音视频资源文件暂停响应时间 |
| 测试用例描述：此用例用来测试播放本地音视频资源时候押下暂停按钮的响应时间 |
| 测试用例编号：TC18 |
| 前提条件：准备好本地音视频资源文件 |
| 测试流程：   1. 使用Chromium浏览器或者基于此系统开发的本地播放器打开准备好的音视频文件资源 2. 播放音视频文件 3. 押下暂停按钮，记录响应时间。 |
| 特殊流程：无 |
| 期望：1秒内进入暂停状态 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：本地音视频资源文件播放品质 |
| 测试用例描述：此用例用来测试播放本地音视频资源的时候，画面是否流程，音质是否合格。 |
| 测试用例编号：TC19 |
| 前提条件：准备好本地音视频资源文件 |
| 测试流程：   1. 使用Chromium浏览器或者基于此系统开发的本地播放器打开准备好的音视频文件资源 2. 观察画面和音质 |
| 特殊流程：无 |
| 期望：画面流畅，音质良好。 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：长时间运行测试 |
| 测试用例描述：此用例用来测试长时间运行过程中，故障发生的概率。 |
| 测试用例编号：TC20 |
| 前提条件：准备好专门的测试服务器 |
| 测试流程：   1. 使用Chromium浏览器打开测试服务器测试页面。 2. 循环播放服务器中的音视频文件 3. 连续待机播放一周，观察故障发生的概率。 |
| 特殊流程：无 |
| 期望：连续播放，不发生死机或播放停止现象。 |
| 测试结果：OK |
| 备注：无 |

|  |
| --- |
| 测试用例名称：内存资源占用测试 |
| 测试用例描述：此用例用来测试正常播放中，占用内存的大小。 |
| 测试用例编号：TC21 |
| 前提条件：准备好专门的测试服务器 |
| 测试流程：   1. 打开浏览器正常播放音视频 2. 查看内存占 |
| 特殊流程：无 |
| 期望：占用内存大小不超过60M |
| 测试结果：OK |
| 备注：无 |

5.3 优化

5.3.1 占用资源的优化

在嵌入式系统中，由于硬件资源十分有限，如果同时打开过个页面进行音视频的播放，则会占用大量的内存空间及处理器的消耗，经过测试，如果同时打开的播放音视频的页面过多，会造成播放卡顿、功耗过大、机器发烫现象。一般来说，在可移动的设备上，不太赞成支持多个音视频文件同事播放，以节省资源。所以，需要对各个页面的播放请求进行管理。

在第4章中我们知道，所有的播放工作都是在Browser进程进行的，而Browser进程在Chromium中是唯一的。所以，每当有播放请求的时候，可以在Browser进程的BrowserMediaPlayerManager对所有播放请求进行管理，控制同时播放的视频的个数。因此，我们可以在新的音视频播放请求加载的时候，暂停之前所有的播放，即每次只播放最新的请求暂停之前的音视频，从而降低资源与消耗。

5.3.2 视频描画效率的优化

当前视频的播放方式是先由解码器材进行解码，然后将解码后的数据存入到一块缓存里，Chromium读取缓存资源进行上屏描画，这是一个跨进程的过程。为了提升效率，可将其改为，由Media Service直接描画，Chromium浏览器在页面上挖洞的方法，偷到下一层的Media Service播放视频的Layer中，在体验上来说，用户看到的依然和之前的效果一样，但是减少了进程间通信的频率，描画效率得到了提升。

5.4 本章小结

本章介绍了本论文的测试及测试结果，是对第2、3、4章整体分析、设计与实现的成果的检查与验证，然后对设计和实现提出了进一步的优化方案。总体来说，系统设计与实现与初中目标一致，基本完成了基于Gstreamer的Chromium音视频播放系统的设计与实现。

6 总结与展望

6.1 工作总结

本文通过对嵌入式设备多媒体数据应用的现状和趋势的了解，结合车载多媒体终端软硬件开发平台技术，对常见构建流媒体应用的开源多媒体框架及浏览器音视频播放体系结构的相关技术进行学习研究。根据开发需求和开源项目Chromium的分析，设计并实现了基于Gstreamer架构的Chromium音视频播放系统。

本课题完成的主要内容包括以下几个方面：

1. Chromium浏览器多媒体播放相关的架构分析与重构设计

Chromium是谷歌开源网页浏览器计划的代号，其核心思想基于高效、稳定、安全等理念，采用苹果公司的Webkit解析排版引擎，附上自主研发V8 Javascript解析引擎，并捆绑了有名的沙盒机制来实现安全访问。由于开源项目的版本更迭频繁，截止目前，Chromium的主版本号己经很多，副版本号更是每时每刻都在改变。对于多媒体播放相关的模块，Chromium本身提供了一个简单的实现方式，但是非常具有局限性，不能共通使用。在资源十分有限的嵌入式车载多媒体终端产品中，不可能为浏览器单独写相应的多媒体应用模块，一般来说，是多个应用共享一个音视频播放动态库，Chromium也调用此共享库进行音视频的播放。因此需要对Chromium音视频播放相关的部分代码进行重构，添加中间层并抽象出比较通用的接口。这一部分的主要工作是：

（1）分析Chromium音视频播放相关的代码结构及实现流程；

（2）根据需求，分析和抽象出中间层用于共同接口的适配；

（3）在原来的基础上重构优化现有的结构和代码。

1. 掌握Gstreamer架构的原理和使用方法

Gstreamer是一个创建流媒体应用程序的框架，在 Linux 系统平台上，通过其管道的设计工作的理念和面向物件的思想，开发者可以设计一个简洁而高效的可扩展型的播放器。这种架构最显著的优点是其提供的功能元件可以有机的接入到各种应用管道程序中，从而降低设计开发工作的复杂性。此外，这种架构是基于插件的，Gstreamer将各种元件和一些基本的功能模块都以插件的形式封装成箱柜，开发者进行开发时可以引用这些功能模块设计特定的处理程序。几个比较重要的基础结构是：元件(Elements)、衬垫（Pads)、管道(pipelines)、箱柜(bin)。

这一部分的主要内容是：研究并设计基于Gstreamer开发框架播放器的后台处理模块架构方案，阐述后台播放通道的创建过程，Gstreamer架构对于事件的处理机制和方法，处理流程等等。

1. Gstreamer音视频播放的实现与接口封装

Gstreamer架构主要包括核心库和插件库，核心库提供了一套开发接口(API)，可以利用这套接口构造特定功能的架构，这种特定功能架构可以利用插件模板封装成GStr­e­a­m­er的插件。Gstreamer架构就好像是一个“组装拼图”，核心库是架构的核心，而基于核心库扩展的功能就好像是一块块的积木。因此，我们可以一用这种模块结构“组装”出播放器的后台处理框架。

这一部分的主要内容是：基于Gstreamer开发框架后台处理模块架构方案编程完成后台模块的应用程序；将Gstreamer的实现用接口封装起来，做成一个共享库，可以被整个车载系统的任一应用程序加载调用，例如本地播放USB存储器中的音乐、视频。这些接口主要包括：播放，暂停，停止，快进，快退，选时播放等等。

1. Chromium浏览器多媒体播放接口与Gstreamer封装接口的适配

在1和3中已经分别对接口进行抽象和封装，这一部分的主要内容就是二者接口的适配与调试，这里考虑采用适当的设计模式，使在使用上更加灵活。还要考虑线程与进程的问题，因为Chromium使用了较多的进程与线程。

最后，根据需求分析提到的各点进行测试验证，基本达到本课题所立的目标。

6.2 问题和展望

目前对于基于Gstreamer的Chromium的音视频播放系统来说，虽然实现了其基目标，但是还存在一些问题和未来亟待完成的工作。

1. 在视频播放过程如果遇见网页嵌套层次过多的情况下，快速滑动当前播放视频的网页会出现视频播放窗口和网页界面坐标位置不能同步的现象，后续可通过修改视频窗口位置坐标的算法进行改善
2. 丰富Gstreamer的解码功能，实现更多音视频格式的解码播放
3. WebRTC（Web Real Time Communication）的实现，即网页及时通讯。

参考文献

[1] 阮胜才.视频点播系统的设计与实现[M].安徽：中国宇航出版社, 2008: 634-635.

[2] 高睿鹏，刘佳玲.基于 FFMPEG 的通用视频插件[D].内蒙古：内蒙古工业大学, 2010, 27(1)：2474-2476.

[3] 赛迪顾问半导体咨询事业部.2005年中国汽车电子市场规模达624 亿元.世界电子元器件，2006年第3期.

[4] 李淑荣.基于Gstreamer的网络媒体播放系统的开发[D]. 中国海洋大学: 中国海洋大学,2012

[5] wikipedia.http://en.wikipedia.org/wiki/GStreamer[J/OL]. last modified on 15 May 2015

[6] Richard John Boulton,Erik Walthinsen,Steve Baker,Leif Johnson,Ronald S. Bultje,Ronald S. Bultje,Tim-Philipp Muller,Wim Taymans.GStreamer PluginWrit-er's Guide ­(1.­5.0­.1­)­[­ J­/O­­L]. http://gstreamer.freedesktop.org/data/doc/gstreamer/head/pwg/html/inde­x.html. 2015.5.21

[7] Wikipedia, http://en.wikipedia.org/wiki/Dynamic-link\_library[J/OL].last modified on 16­ May 2015

[8] Wikipedia, Web Brwoser[EB/OL]. http://en.wikipedia.org/wiki/Web\_browser

[9] Tim Berners-Lee. WorldWideWeb, the first Web client[EB/OL/].http://www.w3.org/People/Berners-Lee/WorldWideWeb.html, 2015-6-28.

[10] 赵经纬,周余,王自强等.基于WebKit的嵌入式浏览器的研究与实现.电子测量技术,200934(3),135-137

[11] 叶卿.嵌入式微浏览器的设计与实现:[D],北京:北京邮电大学.2005,22-25

[12] 西村贤. [EB/0L/].http://www.atmarkit.co.jp/news/200903/30/chrome.html, 2009-3-30.

[13] 朱永盛.WebKit技术内幕[M].北京:电子工业出版社,2014:2-4

[14] 肖梦华.面向智能电视的嵌入式浏览器平台的研究与设计[D].上海:复旦大学.2013

[15] 许晔.发展嵌入式系统是我国后PC时代的战略选择[J].中国科技投资,2010,2:2-3

[16] 梁融凌,余昌嵩. 基于嵌入式Linux电子书的设计与实现[J].牡丹江师范学院学报(自然科学版),2014,4:13-14

[17] 冯礼哲,高声荣,罗怀琴,李逸.发展中的4G网络[J].科技视界,2015,5:23

[18] 罗锐.基于Blackfin平台的嵌入式播放器的研究与实现[D].成都:电子科技大学.2009

[19] 仇昊.基于GStreamer的嵌入式多媒体系统的研究与实现[D].北京:北京邮电大学.2011

[20] Taymans W, Baker S, Wingo A, etal.Gstreamer Application Development Manual (0.10.35.1)

[21] 黄铁军.视频编码国家标准AVS与国际标准MPEG的比较[DB/OL]. http://www.a-vs.org.cn/，2005.

[22] [https://en.wikipedia.org/wiki/Web\_browser, last modified on 7 June 2016](https://en.wikipedia.org/wiki/Web_browser,%20last%20modified%20on%207%20June%202016)

[23] 中国互联网信息中心.中国互联网报告[M].北京.2007

[24] http://trac.webkit.org/wiki/WebKit2. Hosted by Apple. 2016.6.15

[25] <https://gstreamer.freedesktop.org> 2016.8.15

[26] Taymans Wim, Baker Steve, Wingo Andy, et a1. GStreamer Application Development Manual(0.10.33)[EB/OL],https://gstreamer.freedesktop.org/data/doc/gstreamer/0.10.33/manual, 2011-05-10

[27] 张治忠.基于SEP6200处理器的GStreamer媒体播放器优化与实现[D].南京:东南大学.2013

[28] 贺志强.基于Gstreamer媒体播放器的研究与设计[D]. 成都:电子科技大学.2006

[29] 张剑锋.基于GStreamer的STB多媒体播放系统设计与实现[D].上海:上海交通大学.2009

[30] 代坤娟.基于自主Soc媒体播放系统中的音视频同步优化与实现[D].南京:东南大学.2011

[31] 李亚飞.分布式视频转码系统的设计与实现[D].哈尔滨:哈尔滨工业大学.2014

[32] The WebKit Open Source Project [EB/OL]. http://www.webkit.org/

[33] Sten H, Tiger G. Web Operating System for Modern Smart phones[J]. 2011.

[34] 段虎才,倪宏,邓峰等.WebKi t内核的嵌入式浏览器磁盘缓存方法[J].计算机科学与技术.2015,(3):17-19

[35] Google.Multi-process Architecture[EB/OL].http://www.chromium.org/developers/desi­gn­­­-­­­documents­/multi-process-architecture, 2016.9.10

[36] S Alimadadi, S Sequeira, A Mesbah, K Pattabiraman. Understanding JavaScript Event-Based Interactions with Clematis[J]. Acm Transactions on Software Engineering & Methodology. 2016, 25(2):1-38

致 谢

在经过4年的工作实践后，能够重新踏入学校的大门，较为系统的学习软件工程知识，对我来说，实在是难得的机会。在两年多的学习过程中，既得到了良师的教诲，又得到了同学和益友的启发，使我受益良多。

在这篇论文的写作过程中，我的导师姚建国老师给予了我很大的帮助，本论文能够顺利完成，离不开导师的悉心指导和严格要求。姚老师治学态度严谨，工作态度认真，待人和蔼、朴实无华，不仅使我树立了自己的学术目标、掌握了基本的研究方法，还使我明白了许多待人接物与为人处世的道理。本论文从选题到完成，每一步都是在导师的指导下完成的，倾注了导师大量的心血。在此，谨向导师表示崇高的敬意和衷心的感谢！

此外，本文研究工作还得到了上海商泰软件有限公司、杭州微纳科技股份有限公司各位领导与同事的热情帮助，他们在专业领域的通晓给了我极大的帮助，在此向他们表示真诚的谢意！

我还也要感谢学校所有辛苦工作、为我们精心安排每次学习与活动的老师们，感谢他们给予我学业上无私的教诲和生活上亲切的关怀！感谢陪伴我的同学，感谢他们在学习和论文写作过程中给予的帮助和敦促。

同时，尤其感谢多年来一直给予我鼎力支持和无私奉献的父母以及支持和照顾我学业和生活的妻子，感谢他们默默的付出。