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：	Hu Jihao
Student ID:	1130379118
Mentor 1：	Yao Jianguo
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。本课题希望通过对当前网络浏览器的发展规律和未来趋势做简要分析，结合目前车载多媒体的发展现状，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）了解和分析 Gstreamer 基本的框架结构，掌握其设计理念和基本原理。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，讨论存在的不足之处，为下一步的改进工作指出方向和提出目标。

Gstreamer 框架是当前多媒体设计领域较成熟和常用的框架，在此框架下，开发人员可以根据不同的需求灵活地对多媒体格式进行增加，对传输协议的更换也更为便捷。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 web browser and vehicle multimedia, 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) Learn and study of Gstreamer framework and impaortant parts, Master the design concept and basic principles, 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.

Keywords Chromium, WebKit, Multimedia framework, Gstreamer

目 录

1 绪论

1.1 研究背景及意义

在人们的日常生活中使用互联网的场景和频率越来越高，它已成为人们生活的重要组成部分。伴随着计算机通信技术迅猛发展，以及新的多媒体技术不断涌现，之前静态页面和简单的文字早已无法满足消费者对网络信息的需求，人们往往通过网络来获取包含音频和视频的更加丰富多彩的信息。因此，网络多媒体技术，这种对动态图文信息进行加工处理然后直观地呈献给用户的技术，依靠其集成性良好、交互体验友好以及实时性，结合音频和视频的特性[1]，给人们展示了一个色彩斑斓的网络世界，现在网络多媒体技术已经得到广泛的应用，不管是在工作中还是生活中都给人们带来很大便利。

当下，大部分主流浏览器已经对 HTML5 技术和多种音视频媒体格式进行了良好的支持，Flash 和 HTML 页面一般会同时被某些音视频网站使用，尽管现在 Flash 的支持不如以前。Google 公司发行的 Chrome 浏览器可以说是对 HTML5 最铁杆的支持者[2]，它采用了新的渲染引擎架构，并且引入了时下比较流行的新技术。在 Chrome 浏览器上打开音视频文件会感受到不同寻常的流畅性和易用性，而且功耗更低。如果我们能将这些优秀的技术应用于嵌入式设备中，那么将会有其独特的优势。同时，对于我们的应用层软件来说，更易于开发跨平台、高性能、兼容性好的产品。

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，硬件平台和系统平台的差异性和多样性给多媒体系统的开发带来了不小的挑战，开发者往往需要针对不同的平台做定制类的开发。当前多媒体系统的功能也越来越复杂，如果不能妥善解决，对开发成本也造成不同程度的提高。在开发应用程序时候，工作人员需要投入大量的工时去对底层的多媒体处理进行资源整合和恰当使用[4]。

Gstreamer 框架基本的设计思想是基于管道[5]，在这种架构下我们可以方便的添加对新的多媒体格式的支持、以及传输协议的更换等 [6]。我们可以有选择性的把 Gstreamer 提供的功能性元件安装到管道程序中，即插即用，这也是 Gstreamer 框架最显著的优点之一。如果使用 Gstreamer 框架，开发人员可以把多媒体处理相关的处理细节集中管理，并对所有的内部处理进行封装，以接口或者共享库的形式提供给应用程序的开发者，这样上层应用开发者编写多媒体应用程序的时候就不必要再花精力考虑多媒体底层实现的事情，将主要的精力投放到前端的交互体验上面 [7]，以此来节省开发时间，省去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，而且简化了集成和复用，可以快速响应市场需求，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

浏览器是一种软件，它可以把文本样式的 HTML 文档转换成可视化界面，这些文档可以是网页服务器上的，也可以是本地文件系统中存储的，它可以依照一定的规则来解析这些规范性的文档，然后以可视化的图形的形式渲染出来呈献给阅读者，并且可以让用户与这些文件交互和互动。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，现在的浏览器种类已经比较繁多。在全球市场上占有率比较高的主要有微软的 IE、火狐社区的 Firefox、谷歌亲生的 Chrome、苹果公司的 Safari、欧朋的 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 页面扩展功能，它能够用运行在基于 Linux 系统的电视一体机上，同时在机顶盒中也得到了使用。除了具备了传统电视的播放功能外，还具备网页浏览的互联网功能。比如还有谷歌公司推出的 GoogleTV，它是基于 Android 系统的一款娱乐体验系统，以 Android 和 Chrome 为基础，把互联网服务和传统电视服务集合起来，融于一体。GoogleTV 的方案要复杂于 Yahoo ConnectedTV，因为 GoogleTV 除了在系统软件层面做了体系般的定制，还把

硬件相关的东西整合进来，并且包括电视周边外围设备，其目的就是为了使生态系统更加完整。

通常，浏览器都有一个核心的东西，就是所谓的渲染引擎，它负责解析网页文档并把解析结果以图像的方式比较自己管的表达出来。基于渲染引擎来开发浏览器，是常见的做法。浏览器的渲染引擎有很多种，但目前采用的比较多的只有以下几个[10,11]。

Windows 系统原装的浏览器 Internet Explorer（IE）使用的渲染引擎是 Trident。跨平台的、Mozilla Firefox 浏览器的引擎 Gecko，Gecko 对系统的支持比较全面，它不仅可以运行于 Microsoft Windows、而且可以运行于 Linux 系统，就连 Mac OS X 操作系统也支持。Konqueror 是 KDE 桌面系统的一部分，主要用于 Linux 和 BSD 家族的操作系统，其使用的引擎是 KHTML，速度较快，但是容错能力不是太强，对语法的要求比较严格。开源浏览器引擎 WebKit，是目前被采用最广泛的渲染引擎。苹果公司的 Safari 浏览器和谷歌公司的 Chrome 浏览器都采用的是 WebKit 渲染引擎，由于近年来移动互联网的发展以及手机销量的爆炸式增长，这两家公司的浏览器可以说占有了绝对的市场优势，WebKit 自然也成了移动设备中市场份额占有量的佼佼者。就连现在 Chromium 声称的 Blink 渲染引擎，其实本质上还是 WebKit 的衍生品。

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]。

时下对于嵌入式系统来说是一个前景广阔的发展时机，移动互联及物联网的旺盛需求是催生嵌入式市场繁荣的重要因素[15]。随着芯片制造工艺的提升和 CPU 性能的指数级提升，加上最近几年嵌入式系统的新兴发展，越来越多的功能在嵌入式设备上被是实现，个人电脑的一些功能也在逐渐地向嵌入式设备上移植[16]。有些以前需要在计算机上完成的工作，现在也可以在嵌入式设备中完成，比如办公文件的编写以及图形图

像的处理等，都可以很方便的在手机端完成。同时，无线技术的发展也是日新月异，从最初的 2G 到现在的 4G、5G 以及随处可见的 wifi，无线网络的覆盖和速度正在快速的发展，人们已经不再满足于以往单调的媒体服务。近两年迅速崛起的在线直播、网络视频等新兴业务的增长，对嵌入式多媒体的处理能力提出了新的要求。

各种各样的移动网络业务被中国移动、中国联通、中国电信三大运营商不断推出，流量费用的不断降低对移动网络的大规普及起到了强大的推动作用，移动多媒体的市场也越来越大[17]。就当前来说，多媒体应用几乎存在于每一个智能终端设备，它已经成为了不可或缺的功能之一。要想播放音视频文件，多媒体引擎是必不可少的，不管是本地的文件播放还是网络形式的流媒体文件播放，都需要强大的引擎做支撑。只有多媒体引擎足够强大，才能从根本上满足用户的良好的使用体验。

每种嵌入式移动设备上都有各种不同的播放器或者浏览器软件，不同种类的播放器可能对格式的支持不尽相同。目前有各种各样的多媒体文件格式，每种文件格式的编解码方式又有不同。面对如此多的种类和不同的需求，嵌入式设备的多媒体引擎解码库也越来越复杂[18]。同时，解码库依据不同软件平台和硬件平台又有不同的支持，安装也受到一定的限制，也需要许多的技术支持。所以，开发一个播放器用于嵌入式设备中是比较繁琐和复杂的，当然如果是对每一种多媒体文件去编写新的解码程序就会变得工作量巨大，这也是没有必要的。

针对以上繁琐的问题，Gstreamer 提供了完美的解决方案，它的框架简洁，使用简单，思路清晰，开发便捷[19]。工程师们可以使用这种框架结构完成多种多样的音视频处理。Gstreamer 的开发资料是比较少的，对开发来说会有一定的难度，但是它的开发机制比较简洁，结构也比较合理，对于多媒体开发从业人员来说是一个良好的工具，相信随着时间的积累，在 Gstreamer 方向的多媒体开发会变得越来越容易，Gstreamer 也会越来越多的应用在各个领域。

Gstreamer 多媒体应用程序开发框架多流行于 Linux 系统下。它的体系结构是基于元件和管道，在 Gstreamer 框架中几乎可以把所有的功能模块做成可插拔的组件，这些功能模块在需要的时候可以被很容易地安装到目标管道上。利用管道机制可以对几乎所有的插件进行统一的数据管理[20] 。基于以上我们可以看出，使用 Gstreamer 框架可以快速的使用插件拼出一个具有功能相对完整的多媒体应用程序。

提到多媒体，就不得不提到多媒体文件的编码格式。目前国际上已经形成了比较标准的技术规格和协议，这也为促使嵌入式多媒体的发展做出了推动作用。嵌入式系统的资源是十分有限的，微处理器、存储器等都不如个人计算机那么强大，运算能力和

功耗受到制约。而多媒体标准技术的实现对资源的消耗和硬件的要求又是比较高的，这是一个必须要面对的挑战。不过随着 MPEG 系列 H.26X 系列等主要国际标准的硬件级实现，多媒体系统的发展被加快。AVS 是我国具备自主知识产权的第二代信源编码标准，是《信息技术先进音视频编码》系列标准的简称，它在编码效率方面有一定的提高[21]，这个也在某种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、历史记录、设置、下载、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 Render-Object 树直到绘图上下文，如图 2-2 所示的过程。

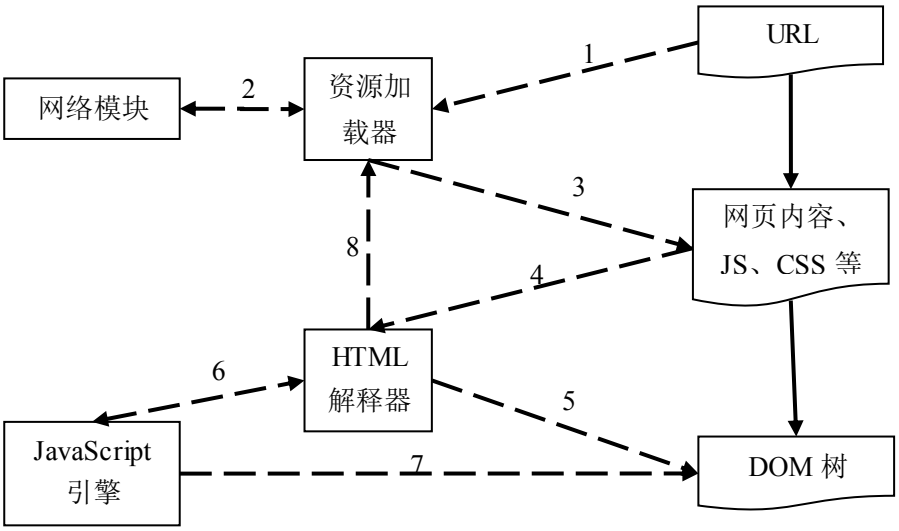


图 2-1 从网页 URL 到 DOM 树
Fig 2-1 The process of generating DOM tree

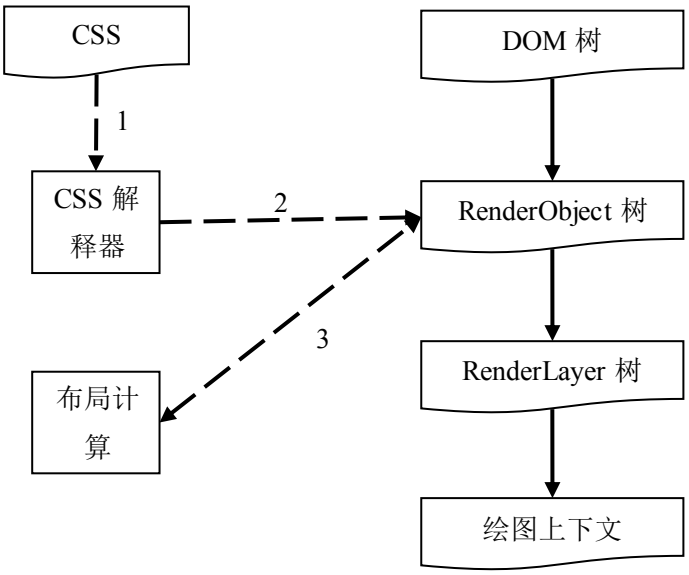


图 2-2 从 CSS 和 DOM 到绘图上下文
Fig 2-2 From CSS and DOM to Drawing context

这一阶段的具体过程如下：CSS 文件被 CSS 解释器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。

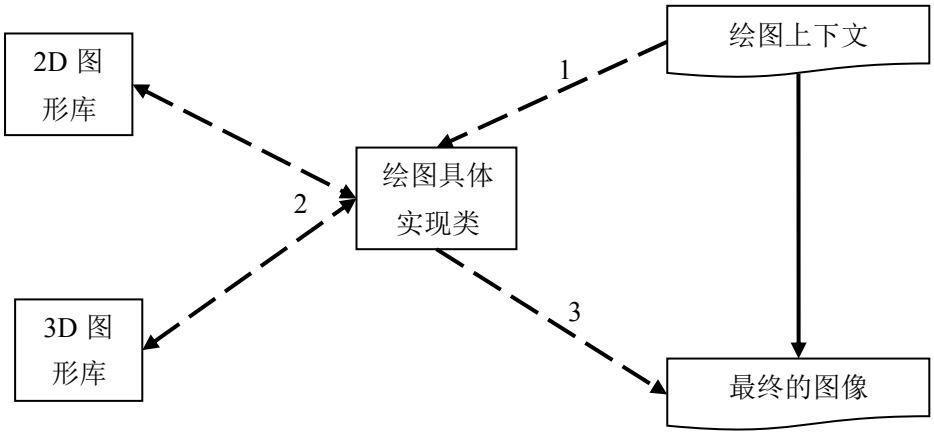


图 2-3 从绘图上下文到最终的图像

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。

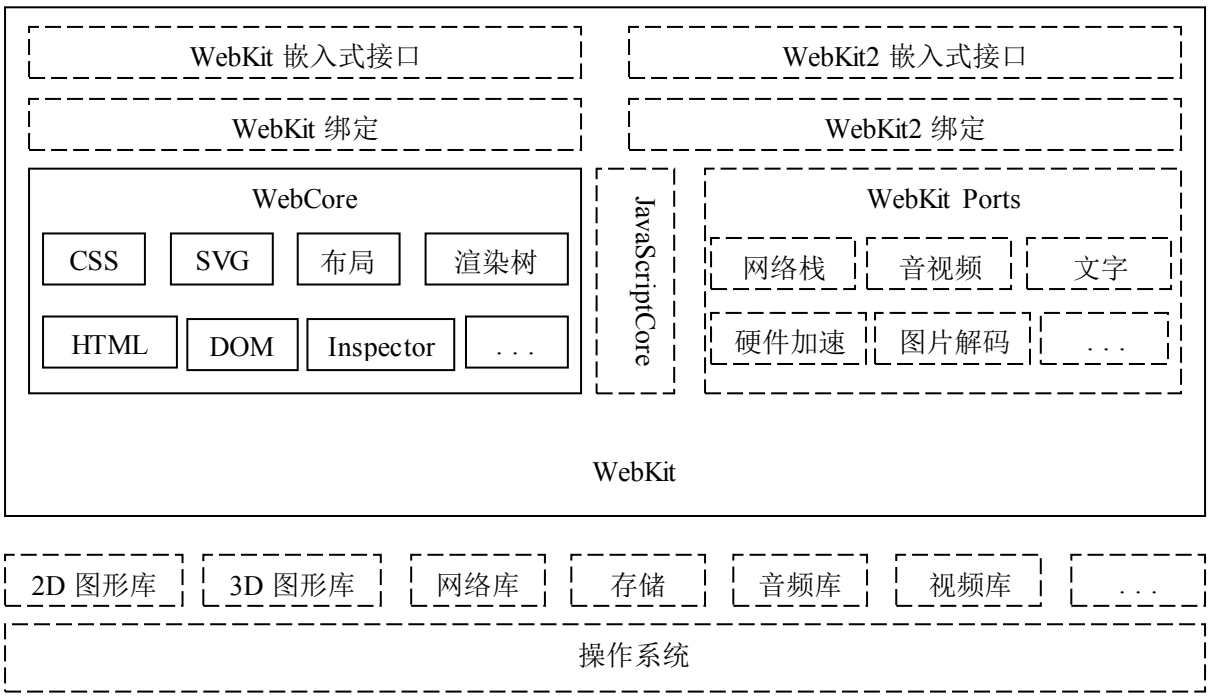


图 2-4 WebKit 架构

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，它的性能不是很好，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。

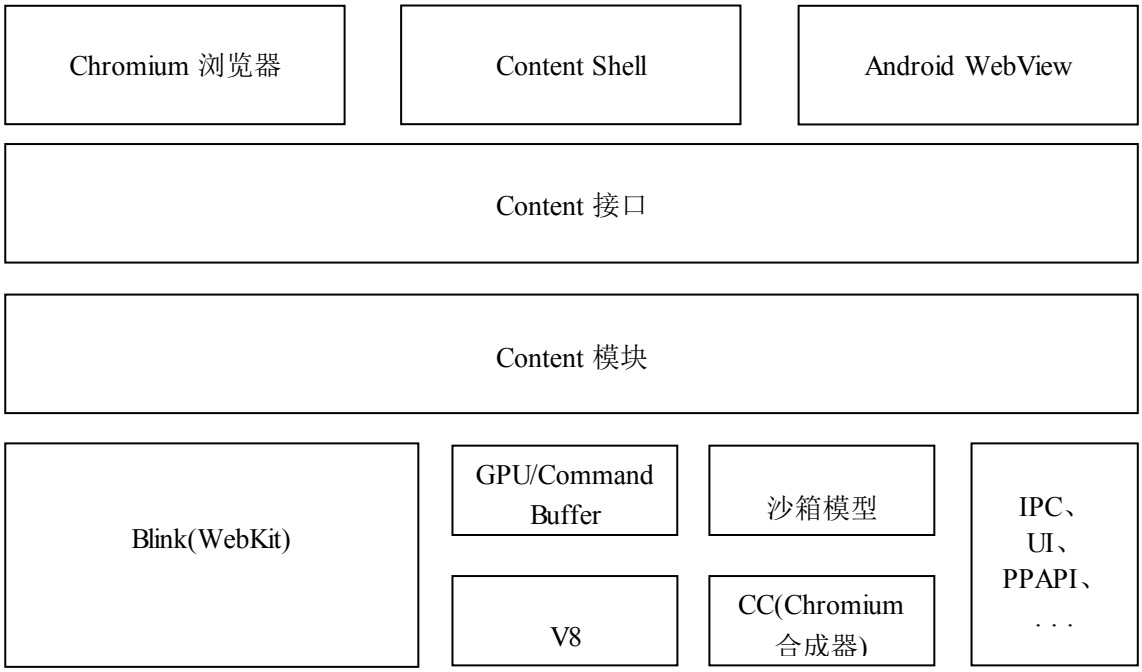


图 2-5 Chromium 模块结构图

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。

2.2 Gstreamer 基础

Gstreamer 是一个多媒体框架库，并且它是开源的。我们可以采用 Gstreamer 方便的开发多媒体模块，比如简单编码格式的音频文件的回放、比如网络流媒体、比如音视频的非线性编辑，等等。解码和过滤技术对于应用程序来说是透明的，上层并不关系具体的实现细节，新的解码器和滤镜可以被开发者使用简单通用的接口编写的插件来添加[25]。Gstreamer 对 DirectShow 的设计思想进行了借鉴，它主要源自视频管道的创意，这些创意是由俄勒冈的一个研究生学院提出的 [26]。插件是 Gstreamer 的重要特征，种类繁多的编解码器以插件的形式提供，当然也有其他功能。在数据流管道中，我们可以任意链接已经提供好的插件。上面说到 Gstreamer 只是一个处理多媒体的框架，在这个框架中插件、数据流、媒体操作被核心库函数调度处理。另外，Gstreamer 的核心库提供了开放给开发者的 API，使用这些 API 开发者可以方便的使用其他插件，以便于快速的开发应用程序。

2.2.1 基本概念

Gstreamer 的基本概念和术语并不是太多，对重要的概念和术语做基本理解是使用 Gstreamer 的前提。下面我们对此进行简单的说明。

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。根据元件类型的不同，它们具有不同的功能，比入 source 表示的是源元件，它只有输出没有输入，只产生数据不处理数据；而 sink 代表的接收元件，它只有输入端没有输出端。这些元件都有函数接口提供，有的用于读取文件数据，有的用于解码数据，有的用于对数据分流，比如 demuxers 就能对视频文件文件中的声音和图像进行分流，将分流出来的数据分别交于声卡和显卡独立处理。我们将这些元件像水管一样对接起来创建一个管道，就能构建一个完整的播放器。很多默认的元件在 Gstreamer 中已经安装了，如果不是构建特别复杂的多媒体处理软件，这些元件基本够用。当然，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）本质上是一段可被执行的代码，在程序运行的过程中根据需要来加载。一般来说会把插件的代码编译成动态库或者是共享库，在程序运行的时候被 load 进去[27]。单个的插件可以只包含一个元件，也可以包含多个元件，我们完全可以把多个元件放入到一个插件之中，使之支持新的功能。在 Gstreamer 中到处都是插件的概念，哪怕只是开发一个简单的多媒体应用。因为 Gstreamer 的基本函数是比较少的，它依靠插件实现诸多的功能。插件在使用的时候会有留下注册信息被保存起来，一般来说是一个 XML 文件，在这个文件中将记载所有注册过的插件的信息。因此，基于 Gstreamer 的多媒体应用程序可以在需要的时候进行插件的加载，而没必要在一开始的时候就把所有的插件全部加载进来。

另外一个重要的基本概念就是衬垫（Pad）。衬垫在 Gstreamer 中起到协商连接的作用，它类似于水管之间的连接头，可以看做是元件之间的连接口，数据在衬垫上流入流出[28]。衬垫具有它自己的特性和独特的数据处理能力，它可以对流经它的数据进行筛选和选择，如果和衬垫所指定的数据类型不符，那么这些数据就无法通过衬垫。比如有解码插件的入口元件上会有一个衬垫来指明自己能够解析的数据类型，当满足于这个数据类型的时候，数据才流向这个插件。也就是说如果一个衬垫的数据想要流向另一个衬垫，那和这两个衬垫所指定的数据类型必须是一致的。

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）

源元件处于整个管道的最前面，它负责从磁盘或者其他存储介质中将原始的数据读出来，是数据的产生者。与其他元件不同，source element 有且只有一个源衬垫，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。

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

这两类过滤元件都具有输入端与输出端，过滤器和类过滤元件对从输入衬垫流入的数据进行处理，完毕后经由输出衬垫将处理过的数据留给下一个元件。这类的元件种类是比较多的，比如常见的音量元件，以及 ogg 分流器，或者视频转换器、还有部分编码格式的解码器，等等。类过滤元件能够拥有一到多个的元衬垫和接收衬垫。比如视频分流器，它或许会拥有一个接收衬垫，但是可能会有多个源衬垫。而解码器一般来说只会拥有一个接收衬垫和一个源衬垫。

3. 接收元件（sink elements）

接收元件是管道的“终结者”，处在管道的最末端，它仅有接收衬垫。因为它是不产生数据的，只用来接收数据。比如使用显卡显示图像或者视频，通过声卡出声，还有写磁盘操作，这些首饰由接收元件实现的。

4. 箱柜（bins）和管道（pipelines）

箱柜是可以看做是一个容器，里面可以装载各种元件。管道可以看做箱柜的一个特殊子类型，包含在管道内部的所有元件都能被管道操作。使用箱柜可以降低程序开发的复杂度，我们可以像操作元件一样操作箱柜，因为箱柜本来就是元件的子集。如果我们改变了箱柜的状态，那么箱柜内的所有元件的状态也将被改变。箱柜可以给它的子集元件发送总线消息（bus messages）。管道是一个高级箱柜，如果一个管道处于停止状态，给它设成播放状态，那么数据就开始在整个管道中流动，此时多媒体数据的处理也同时开始了。一旦开始，Gstreamer 会使用一个独立线程承载和维护管道的运行，当数据处理完毕或者状态被设置成停止的时候，运行才会结束。

2.2.2 总线（Bus）

我们可以把总线看成一个简单的系统，它自己有一套独特的线程机制，使用管道的这套线程机制可以把消息方便的分发到应用程序中。当也正是因为管道拥有这一套线程处理的机制，在我们基于 Streamer 进行多媒体应用程序开发的时候，就不用太过关

注线程的区分和识别问题，不管 Gstreamer 是加载了一个线程还是多个线程，我们只需要专注于监听总线消息就好了。

管道被创建的时候，有一个总线被默认包含。所以对于开发者来讲，就不用再创建总线，给总线设计一个消息处理器就好，当主循环 run 起来的时候，总线会不断的查看消息处理器，看看有没有消息需要处理，如果总线发现有新的消息发来并且需要处理，那么总线便立即调用相关的回调函数进行处理。

有两种使用总线的方法[29]。开启 GLib/Gtk+的 main loop，然后设置监听器对总线进行消息侦听，这是第一种方法。如果使用这种方法，Glib 的 main loop 将轮询总线上是否有新消息发来，如果侦听到新的 message，总线会立即发出通知。关于这种 case，有两个函数必不可少，就是对监测进行添加的函数 gst_bus_add_watch()以及 gst_bus_add_signal_watch()两个函数。如果 pipeline 发出一个 message 到 bus，就会触发这个 message 处理器，message 处理器就着手分辨消息信号种类以此判断哪些将处理哪些事件。如果不采用前边讲到的第一种方法，那么还有一种方法可以使用，那就是自己对总线消息进行监听，这会用到两个函数 gst_bus_peek()和 gst_bus_poll()，此处不再展开。

对于总线传递的消息类型，在 Gstreamer 中有几种常见的是事先被预定义的。当然，开发者也可以扩展这些消息类型，根据具体的工作需求在插件中定义新的消息类型[30]。这些消息中都会携带 info 以标识每个信息，比如消息来哪里、消息的类型、以及消息发出的时间戳。如果我们想知道某个消息的消息源在哪里，通过读取 info 就可以识别出此消息出自哪个 element。

2.2.3 Gstreamer 使用方法

下面，我们以一个真实的案例来说明如何基于 Gstreamer 框架的组件创建一个功能不那么复杂的 MP3 播放器。在这里我们假设 MP3 音乐文件存储在本地磁盘上，source element 从磁盘中将该 MP3 音频数据读取出来，然后使用 filter element 对数据进行 decode，最后经 sink element 将解码后的数据写入声卡。

首先我们在 main 函数中调用 gst_init()来进行初始化工作，以便 Gstreamer 函数库能够都到用户传递进来的参数。一个经典的基于 Gstreamer 的多媒体处理程序的初始化过程可参看如下代码[31]：

```
#include < gst/gst.h >
```

```
int main ( int argc, char *argv[] )
{
    gst_init( &argc, &argv );
}
```

然后我们创建三个元件并把他们连接成一个管道。GstElement 是 Gstreamer 的所有元件的基类，因此我们在声明各类型元件的时候都可以使用基类的数据类型：

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

前面说过 pipeline 在 Gstreamer 框架是特殊的箱柜，可以用来容纳和管理元件，现在我们创建管道，用变量 pipeline 来引用我们的管道：

```
pipeline = gst_pipeline_new("my-new-pipeline");
```

source element 是数据的产生者，他有一个属性叫 location，可以把准备好的 MP3 音频文件所在的路径设置给这个属性。如下 filesrc 是创建的数据源元件，argv[1]表示用户输入的 MP3 文件路径：

```
filesrc = gst_element_factory_make( "filesrc",  "disk_source" );
g_object_set( G_OBJECT(filesrc), "location", argv[1], NULL );
```

创建 filter element：

```
decoder = gst_element_factory_make ("mad", "decoder");
```

创建 sink element：

```
audiosink = gst_element_factory_make("audiosink", "play_audio");
```

三个元件已经创建完毕，它们依次是源元件 filesrc，过滤器元件 decoder，接收器元件 audiosink，下面使用函数 gst_bin_add_many 把这三个元件添加到管道中，并使用 gst_element_link_many 把他们按顺序连接起来：

```
gst_bin_add_many(GST_BIN(pipeline), filesrc, decoder, audiosink, NULL);
gst_element_link_many(filesrc, decoder, audiosink, NULL);
```

以上，便完成了基本的准备工作，下面就是直接通过切换管道状态的方式，开始整个流程，我们把状态切换成 PLAYING：

```
gst_element_set_state(pipeline, GST_STATE_PLAYING);
```

若是要监听总线消息，可调用函数 gst_pipeline_get_bus(GST_PIPELINE(pipeline))获取到 bus，然后通过 gst_bus_add_watch 添加总线消息的监听，回调函数是 callback：

```
gst_bus_add_watch(bus, callback, loop)
```

callback 可以定义如下：

```
static gboolean bus_call(GstBus *bus, GstMessage *msg, gpointer data)
{
    // TO DO
}
```

播放结束的时候，我们将状态设为 NULL 终止管道，然后接触对 pipeline 资源的引用：

```
gst_element_set_state(pipeline, GST_STATE_NULL);
gst_object_unref(GST_OBJECT(pipeline));
```

至此，利用 Gstreamer 创建一个简单的 MP3 播放器示例介绍完毕，其过程并不复杂，其目的是帮助我们理解 Gstreamer 基本概念和熟悉它的使用方法，为后文的展开做铺垫。

2.3 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。

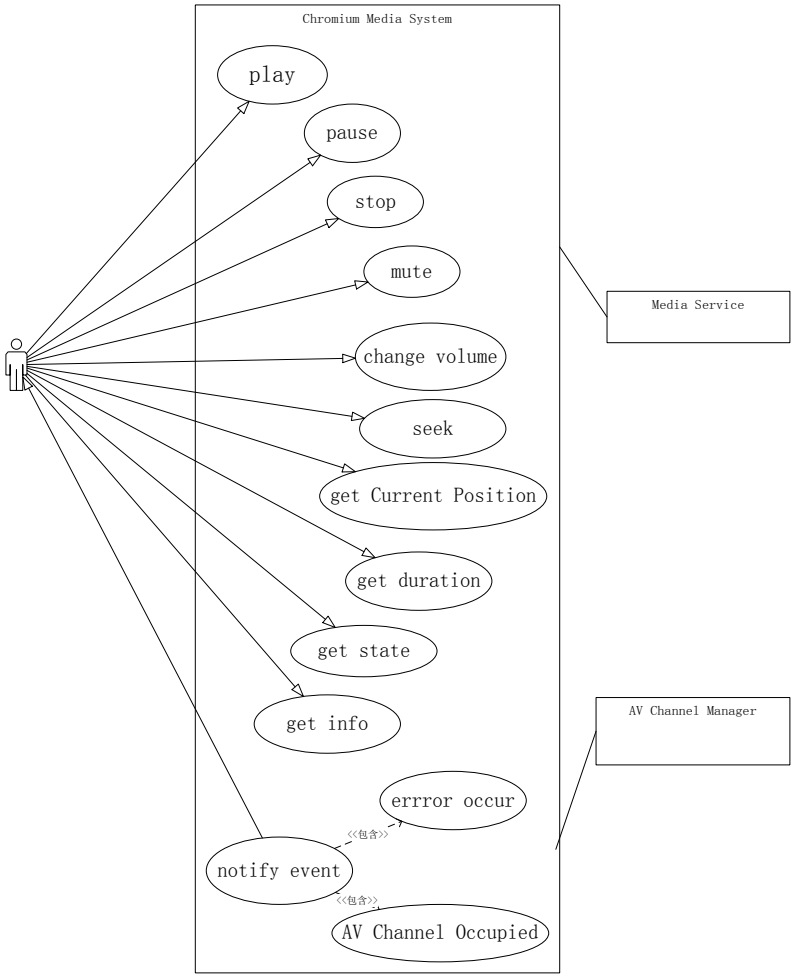


图 3-1 管理控制用例图

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 2. 用户点击播放按钮 3. 页面开始加载音视频文件 A2：网速较慢 4. 当数据加载满足播放条件，会看到播放开始 5. 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 3. 再次点击静音按钮，音量恢复正常 4. 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 2. 点击播放按钮进入到播放状态 3. 点击暂停按钮进入到暂停状态 4. 点击停止按钮进入到停止状态 5. 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。

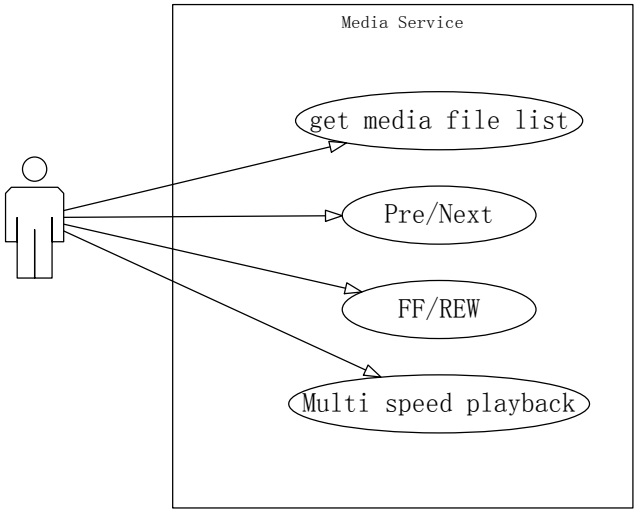


图 3-2 对传统播放器的支持用例图

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 3. 播放列表里的上一个或下一个音视频文件开始播放 4. 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）从发出播放请求到音视频文件开始播放，其时间间隔不超过 2 秒。

（2）从发出暂停请求到音视频文件播放状态切换为暂停，其时间间隔不超过 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

依照规范的开发流程，编码前要首先完成基本设计，程序的结构设计要合理，对模块进行有效的封装，增强内聚性，耦合程度要低。代码的可读性强，接口定义规范。代码要易于维护，易于扩展、可重用性强、移植性好；注意代码的执行效率和运行时内存空间的占用。

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。

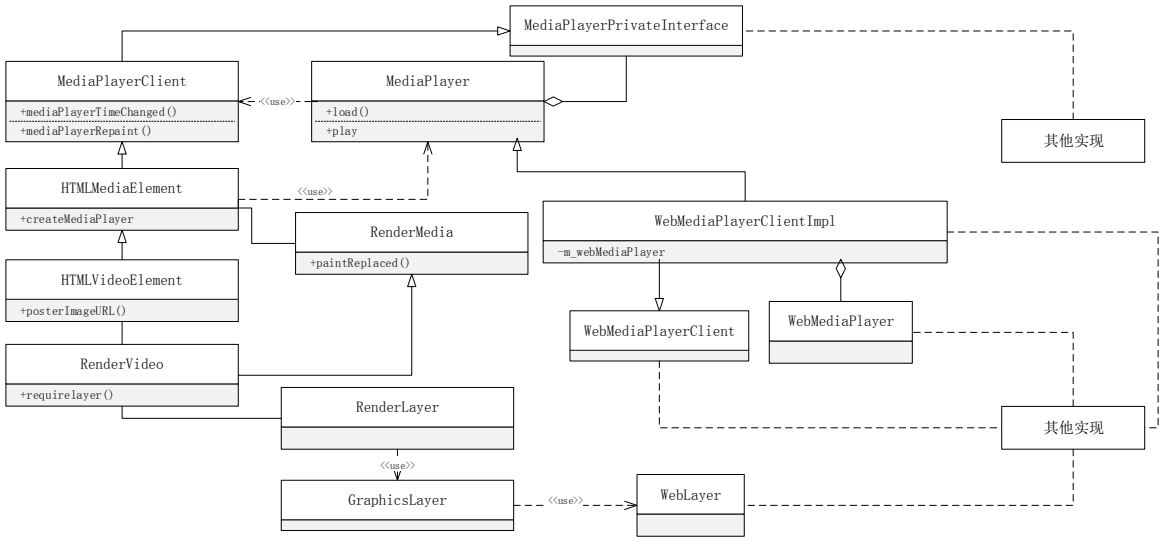


图 4-1 WebKit 支持视频的基础类和关系

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。

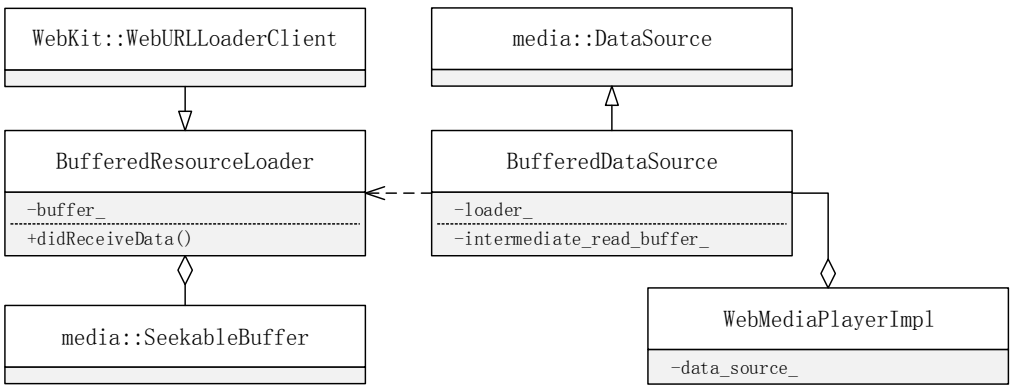


图 4-2 Chromium 的视频资源缓存类

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。

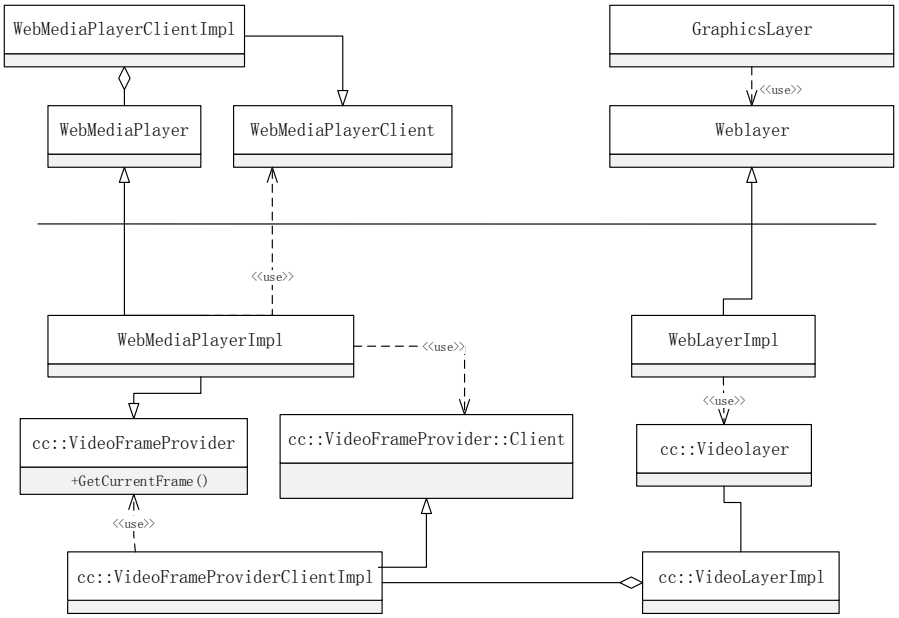


图 4-3 Chromium 支持视频的基础类和关系

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，WebMediaPla

yerImpl 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 4-4 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。

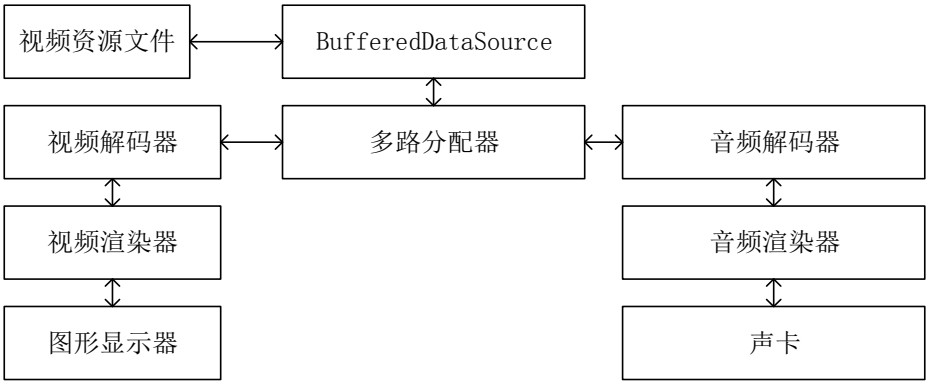


图 4-4 Chromium 的多媒体管线化引擎

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：

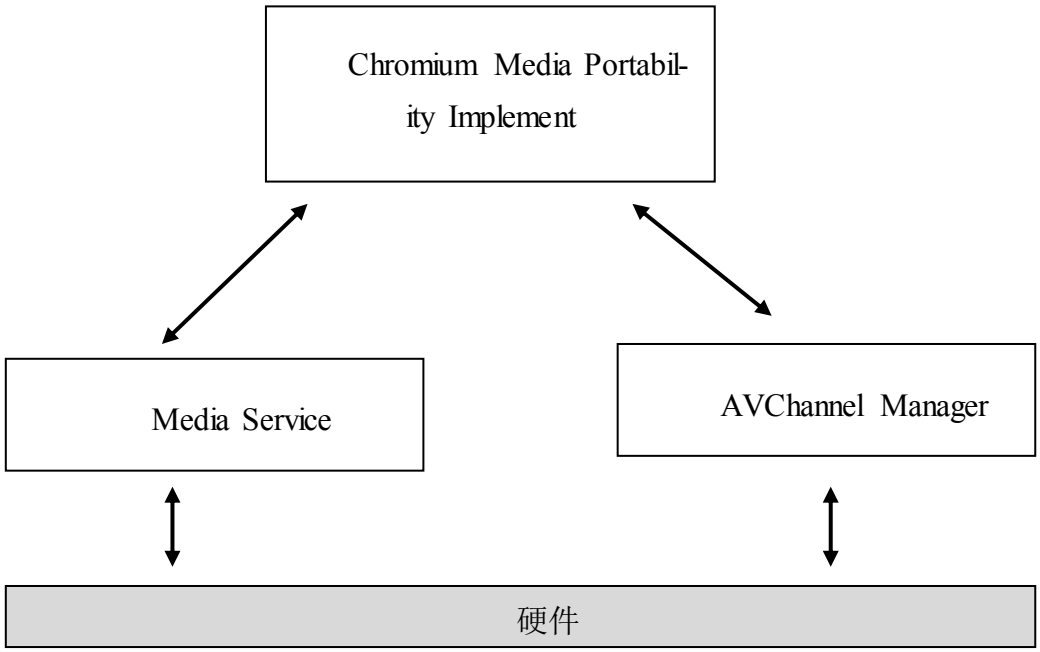


图 4-5 整体模块划分

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。

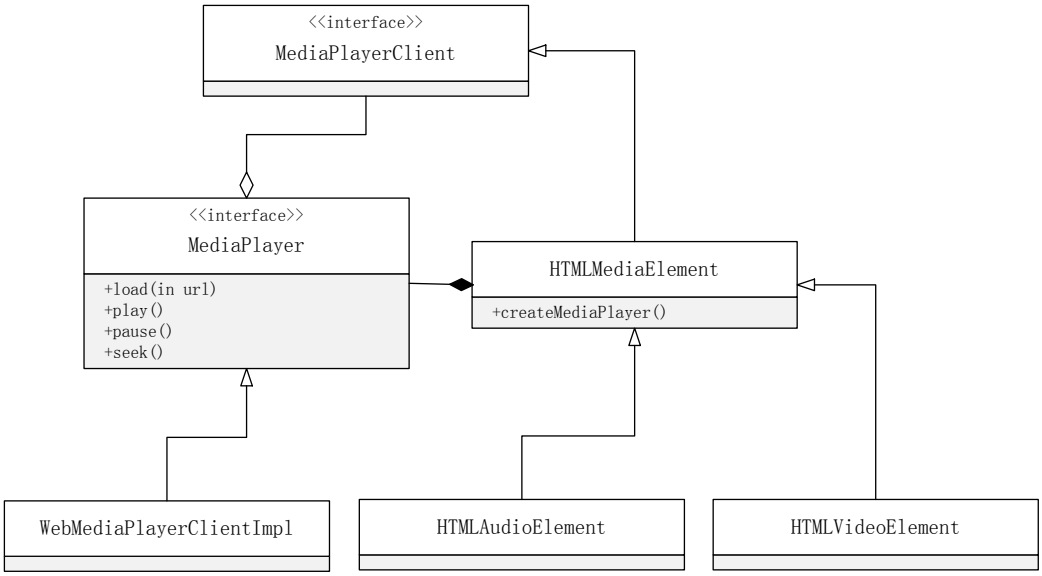


图 4-6 Chromium 中媒体类派生关系

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 WebMediaPlaye-rClient 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。

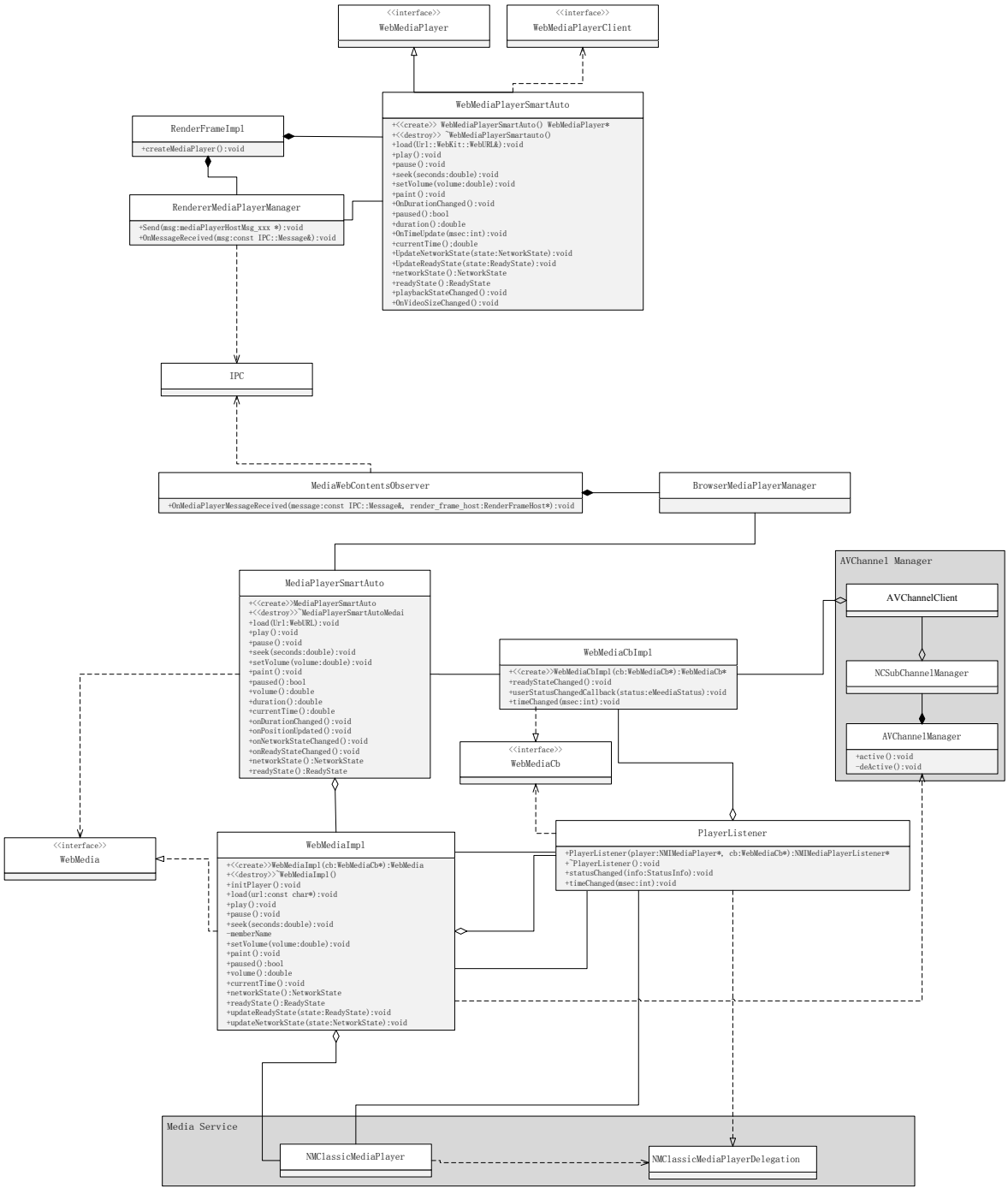


图 4-7 Chromium Media 的实现

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	
处理概要		加载文件	
参数	1	WebURL	网络音视频资源文件的路径
返值		void	

関数名称		play	
处理概要		播放	
参数	0	void	
返值		void	

関数名称		pause	
处理概要		暂停	
参数	0	void	

返值	void	

関数名称		paint	
处理概要		描画视频	
参数	1	Rect	描画区域
返值		void	

関数名称		paused	
处理概要		判断是否暂停	
参数	0	void	
返值		bool	true:暂停 false:非暂停

関数名称		currentTime	
处理概要		获取当前播放进度	
参数	0	void	
返值		double	当前播放时间,单位 sec

関数名称		duration	
处理概要		获取音视频文件总时长	
参数	0	void	
返值		double	音视频文件总时长, 单位 sec

関数名称		OnTimeUpdate	
处理概要		进度更新通知	
参数	1	double	当前进度
返值		void	

関数名称		OnDurationChanged	
处理概要		总时长更新通知	
参数	1	double	音视频文件总时长
返值		void	

関数名称	UpdateReadyState
处理概要	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	
处理概要		获取准备状态	
参数	0	void	
返值		ReadyState	准备状态的定义参看函数 UpdateReadyState 中 ReadyState 的定义

関数名称		UpdateNetworkState	
处理概要		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	
返值		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	
返值		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。

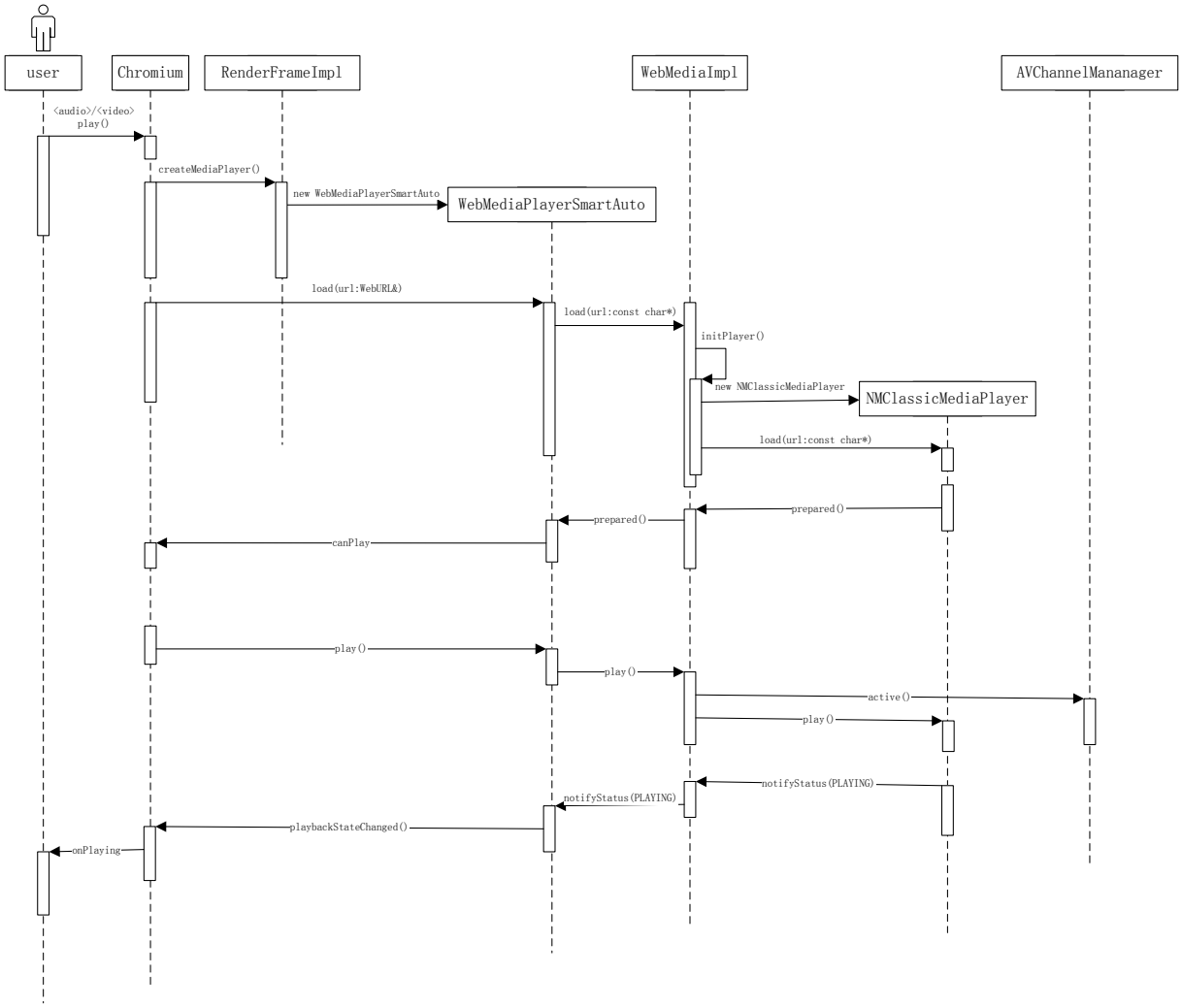


图 4-8 播放器从被创建到开始播放的序列图

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

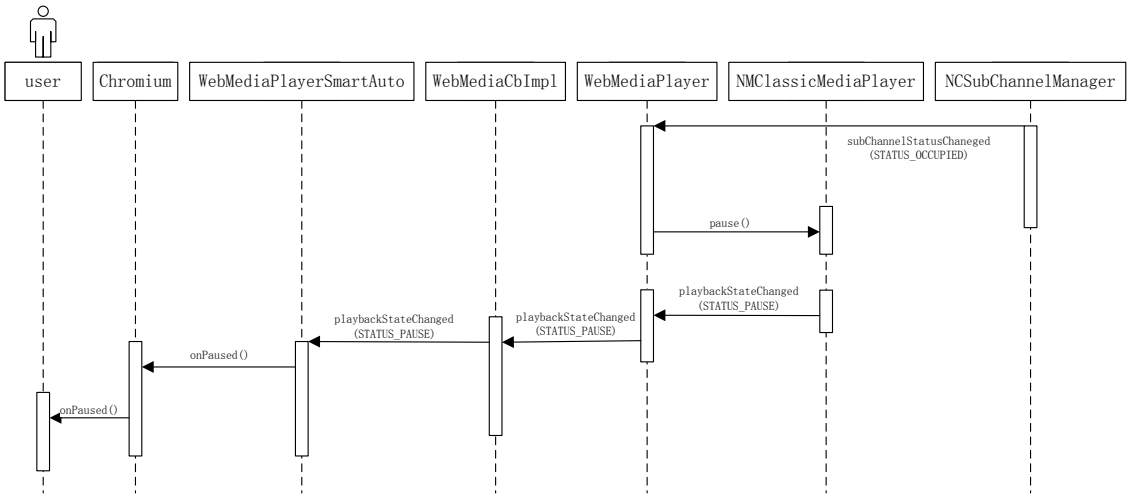


图 4-9 声音通道被抢占的序列图

Fig4-9 The sequence diagram of the sound channel 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，就使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。

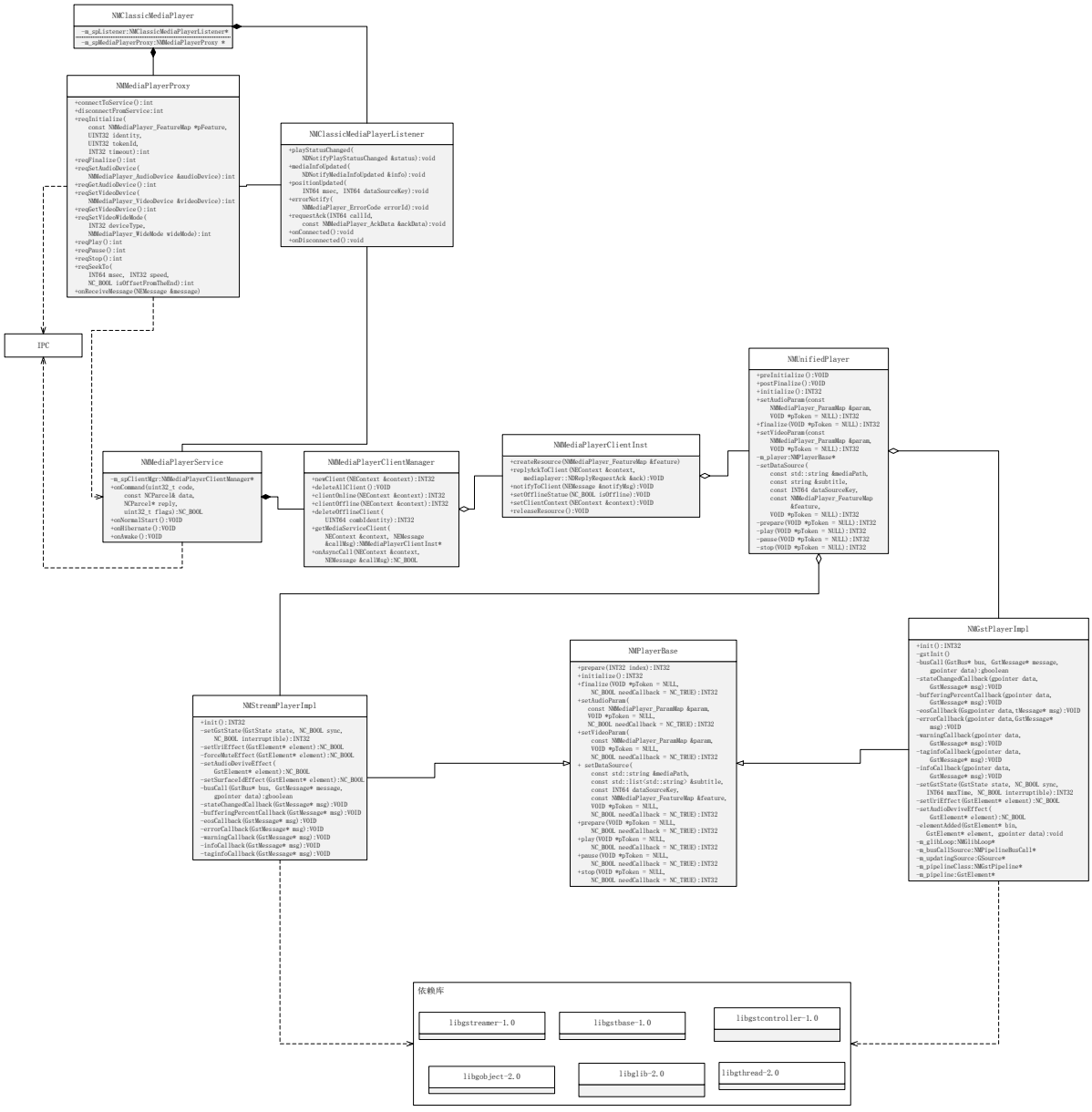


图 4-10 Media Service 类图

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。

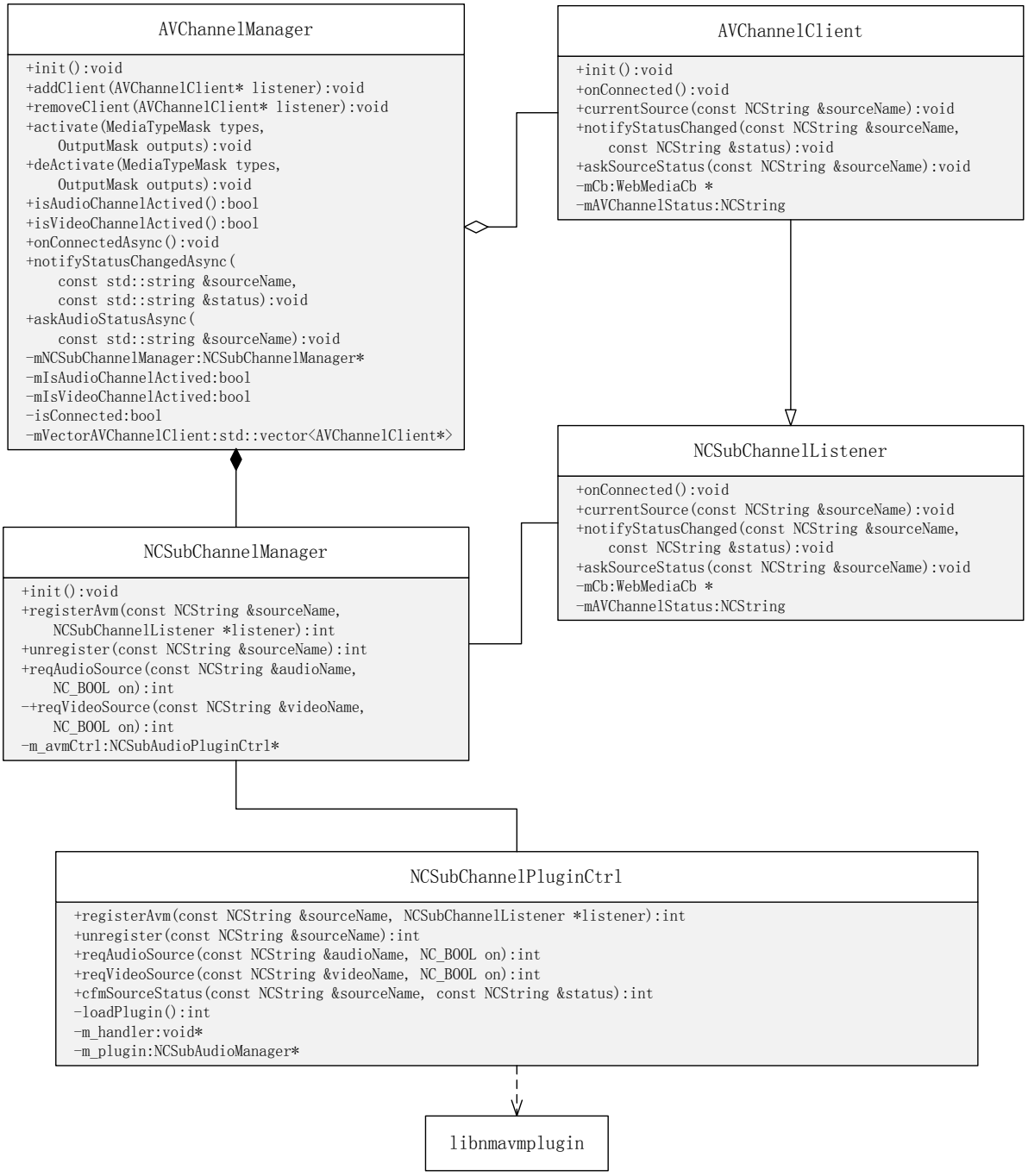


图 4-11 AVChannel Manager 类图

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，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官方网站，里面以形象的图文形式对 Chromium 的多进程架构做了直观的说明。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->getGlib-
Context(), (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 NMGst-PlayerImpl 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(), (GSource-
Func)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 AVChannel-Manager 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。

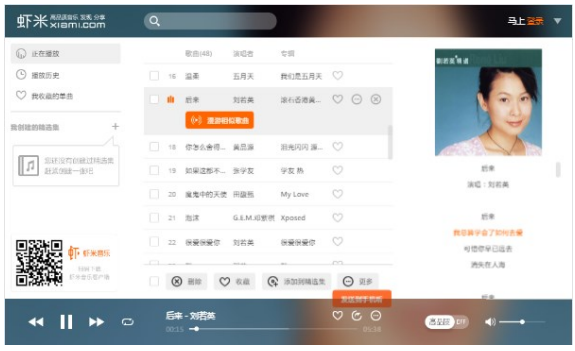
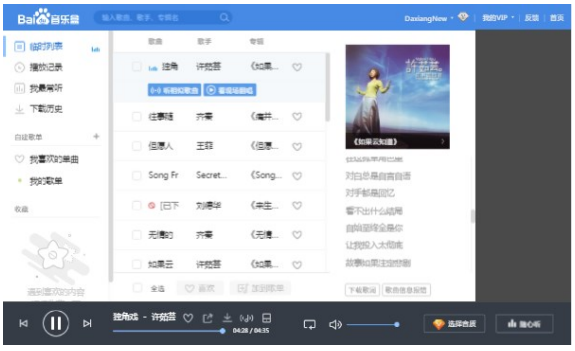
4.4 本章小结

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。

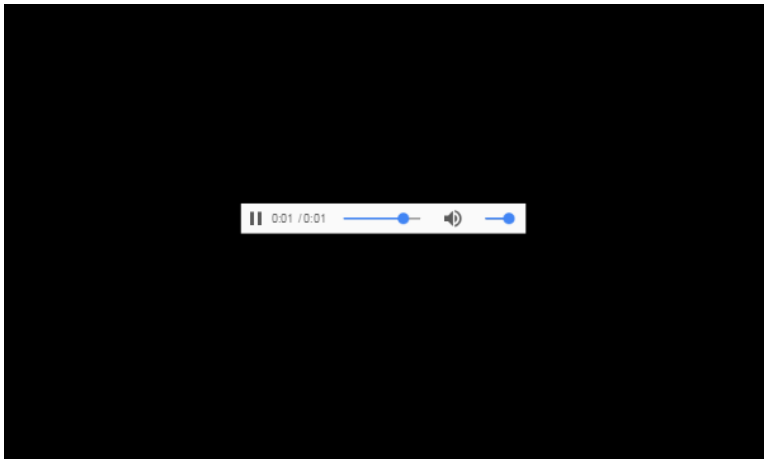
5 测试与优化

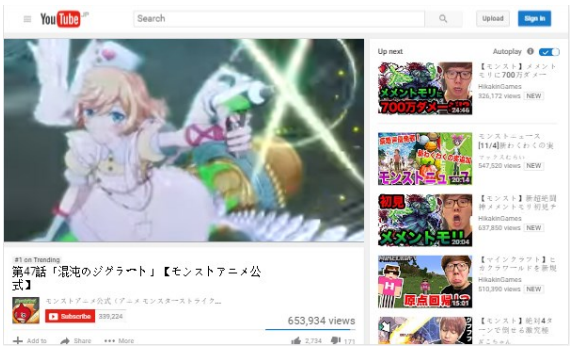
5.1 功能测试

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时播放，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进行解码，然后将解码后的数据存入到一块内存里，Chromium 读取内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，透到下一层的 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 是谷歌公司发布的开源浏览器，具有高效、稳定等特点，它的渲染引擎选用的是主流的 WebKit，由于采用了沙盒机制，所以其安全性得到保证。Chromium 是一个开源项目，任何组织和个人都可以在其工程的官网上获取源代码。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通接口的适配；

（3）在原来的基础上重构优化现有的结构和代码。

2. 掌握 Gstreamer 架构的原理和使用方法

Gstreamer 是一个多媒体开发框架，基于此框架，开发者能够快速的开发出满足需求的多媒体应用程序。其提供的功能元件能够巧妙的接入到任何应用管道程序中，是这种框架最吸引开发者的优点，这种优点可以使设计和开发的复杂度大大降低。Gstreamer 是基于插件的，它可以把多个元件封装起来组成一个具有特定功能的模块，以插件的形式提供给第三方使用，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)。Gstreamer 通过把若干 elements 链接在一起构成 pipeline 实现对媒体内容的处理，element 通过 plug-in 的方式提供。bin 是一种特殊的 element，是由多个其它 elements 组成的。Gstreamer 的核心实现了对 plug-in 的注册和加载等功能，可以通过编写 plug-in 方式对 gstreamer 的功能进行扩展，包括编码方式，封装格式等各种功能。

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。

3. Gstreamer 音视频播放的实现与接口封装

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。

4. 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 cli-ent[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

[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.WebKit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

阔别校园已经 4 年，而今有机会再次步入大学殿堂开始一段新的学习生活，我感到特别的荣幸和感恩。在这几年的时间里，我不仅系统地学习了软件工程相关的知识，而且丰富了分析问题的方法和思路。更让我印象深刻的是敬爱的老师们的谆谆教诲、同学们真诚的友谊。真的非常感谢，感谢良师益友，感谢陪伴的每时每刻，感谢无法追回的岁月，感谢留下的美好回忆。

在这篇论文的完成过程中，我的导师姚建国老师给予了我很大的帮助，从选题到开题报告，从写作纲要到认真地指出每版存在的不足，严格把关，循循善诱，在此表示衷心的感谢。姚老师治学态度严谨，工作态度认真，待人和蔼、朴实无华，不仅带领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。本论文从开始到完成，每一步都离不开导师的指点和陪伴。再次，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。

攻读学位期间发表的学术论文目录

[1] 胡济豪，Android 系统智能电视 HAL 层 Sensor 数据传输的一种实现[J/OL]，上海交通大学软件学院内部网站公示，2016.12