基于 WebAssembly 和 Electron 的跨平台网络代理系统研究与实现

学号：205544

学生：缑通旺

指导老师：孔佑勇

目录

一、选题依据与工程应用价值

1.1 选题依据

在网络高度普及的时代，我们常使用的电脑、手机、平板还有 IOT、智能家具设备等等，越来越多的生活物品与网络连接，而接入网络也代表着设备在无时无刻的请求服务，处理返回结果。网络代理系统则是通过拦截计算机上的网络请求，将捕获到的请求根据计算机不同层的网络协议解析成不同的报文信息，再进一步对报文信息进行分析和统计并反馈给使用者的工具应用。

网络代理工具通常是软件开发人员和运维人员开发、调试、排查线上服务所需要使用的生产力工具。在软件开发过程中，开发、运维人员往往要面临多个存在差异的环境，针对不同环境的测试、开发一致性是最大的问题，也是影响开发效率的关键原因之一。网络代理的实现方式目前主要有两种，一种是通过直接监听计算机上操作系统的网卡，拦截所有经过网卡的数据帧，再根据网络协议树解析为对应协议的报文信息，从而实现对网络数据包在产生和运输过程中的代理能力。第二种方式是通过启动一个代理服务，将计算机上的请求交由代理服务完成，这样代理服务就可以获取到计算机上所有经过其代理的网络请求。

跨平台桌面应用的实现实际上是利用 Web 技术开发桌面应用，结合 Web 技术的丰富多样性、跨平台的优势，改变传统桌面开发周期慢、效率低、复杂度高等等问题。跨平台桌面应用开发利用 NodeJS 丰富的 API 环境和 Chromium V8 引擎带来的运行效率，使得 JavaScript 在实现页面逻辑的同时又可以和操作系统交互。但是同时 JavaScript 相比较 C++的执行效率低，打包文件过大的问题却是此类跨平台技术开发大型复杂桌面应用的瓶颈，本文希望通过结合 WebAssembly 二进制高性能编码来解决上述问题。WebAssembly 简单来说，就是将 CPU 密集型计算消耗性能部分交还给 C++等底层语言，通过编译为二进制编码内嵌到 JavaScript 运行时，通过虚拟机来实现 JavaScript 执行 WASM 编码的能力，从而解决瓶颈问题。

关于网络代理的解决方案，常见的网络代理工具功能页面不够丰富、不支持跨平台、无页面可操作性差、复杂度高等等。因此通过毕业设计想要实现一个基于 NodeJS 和 WebAssembly 的 HTTP/HTTPS 桌面网络代理系统，通过代理请求的能力自动化管理应用开发生命周期中接口的迭代以及提高对不同开发环境下的兼容能力，致力于让开发人员能够精确的掌握自己的开发环境，通过代理使用规则转发、修改每一个请求和响应的内容；同时通过对包信息解析反馈给开发人员，提升开发人员的开发效率和编程体验。

1.2 工程应用价值

随着云时代的来临，线上环境更加斑驳繁杂、应用系统的生命周期管理等都会面临更大的挑战，通过网络代理工具可以更加智能化应对这些挑战。软件开发生命周期内，应用功能的迭代、上线，需要经过很多环境，例如本地环境、测试环境、预发布环境、正式环境、分地区环境等等甚至更多。每个环境又有其差异性和特殊性，而开发中经常需要在本地开发连接不同的线上资源环境进行测试、开发、调试，频繁切换开发连接环境无疑是一件重复的、低效率的操作。

对于程序开发人员，通过网络代理工具可以高效的切换连接的环境资源，更加准确的掌控当前的开发环境，让我们更加专注于功能的开发研究，而不会过多承担环境等因素带来的心智负担。开发环境是影响开发效能的最大因素之一，不可用的环境、不稳定的上游环境、线上本地环境的差异等等，都会严重影响开发体验与效率。同时通过网络代理解析可以自动化的帮助管理开发应用过程中的接口迭代，可以帮助我们分析当前程序中的接口依赖程度、接口的稳定性、接口的迭代等等，保障应用的生命周期管理。对于运维人员，网络代理工具可以帮助排查线上问题，解决线上问题难查、难以复现的问题。对于普通用户，网络代理工具可以帮助你可视化你的浏览历史，屏蔽非法请求。

二、国内外研究现状与发展动态

本小节将从跨平台桌面应用技术、WebAssembly 技术、网络代理技术三个方面分别介绍其相关的发展动态。

2.1 跨平台桌面应用技术研究

传统的桌面应用程序开发在一段时间内主导了软件开发的进程，随着 Web 应用程序的发展，传统的桌面应用开发因为其不够灵活、组件审美不丰富、迭代周期慢、复杂度高等缺点导致传统的桌面应用程序也在吸收着 Web 技术的优势，产生了跨平台的桌面应用开发技术。传统的桌面应用开发是强依赖于不同的操作系统底层架构，所以在不同的操作系统上都有特定的开发框架和开发语言。桌面应用需要更加关注与操作系统的交互以及多线程的使用，需要严格系统文件权限、系统托盘、剪贴板以及系统软硬件的的差异性。

如表一所示，Windows 操作系统和 MacOS 操作系统几乎占据了 90%多的市场份额，导致大部分开发框架的发展由这两种操作系统主导，而 Linux 操作系统通常用作服务器，对桌面应用的需求不高。

表一 操作系统市场份额

操作系统	市场占有率
Windows	74.27%
MacOS	16.05%
Linux	2.09%

表二 桌面开发框架详情

框架/类库	语言/环境	适用平台
QT	C/C++	跨平台
WPF	.NET/C#/VB	Windows
WinForm	.NET/C#	Windows
MFC	C/C++	Windows

Swift	Objective-C	Mac
Swing/JavaFx/SWT	Java	跨平台
NW.JS	JavaScript	跨平台
Electron	JavaScript	跨平台

如表二所示，我们可以简单将桌面应用开发分为三个方面：

第一，原生桌面应用开发

直接将程序编译为目标平台的二进制可执行文件，调用系统 API，完成界面绘制。Windows 平台主要是 C#、.NET 语言框架，从早期的 MFC 方案到 WPF 和 WinForm,而 MacOS 系统则主要是 Objective-C 语言和 Cocoa 框架。但是原生桌面应用开发的缺点就是，不跨平台、复杂度高、迭代周期长，但是往往运行效率对操作系统的结合性都非常高，更适合大型复杂的桌面应用开发。

第二，QT、JavaFx 一类的跨平台的框架

首先 QT，是一个 GUI 的框架，支持跨平台，易移植，语法结构简单清晰，相比较原生更加容易简单。而且 QT 不仅仅支持 C++，同时也支持 Python 等。但是 QT 学习成本比较大，涉及到协议、QML 等，开发周期也会比较长。但是其开源、丰富的 UI 库和文档生态，也是很多桌面客户端的首选。

以 Java 为首的 JavaFx、Swing 也是一类比较重要的开发模式，优势是和 Java 的天然结合，但是其生态较差，并且 Java 运行时还要通过 JVM 来管理和维护 Java 类对象的内存分配，性能上不如 C++等，开发周期又不如 Web，同时组件也相对较少。

第三种，Web 桌面应用开发

Web 技术开发桌面应用程序，从早期的 node-webkit 到 NW.js 在到如今的 Electron，目前有很多桌面应用都是基于 Electron 框架开发，例如 Visual Studio

Code、Atom、WordPress 等等。随着 Chrome V8 引擎的出现，提高的 JavaScript 的执行性能，让 JavaScript 可以承担更复杂的应用/开发。

Electron 是一个基于 Web 构建桌面应用程序的底层工具框架。 它允许使用 Node.js 和 Chromium (V8 引擎内核库)完成桌面 GUI 应用程序的开发。通过嵌入 Chromium 和 Node.js 到二进制的 Electron 可以构建跨平台桌面程序。为了提供原生系统的 GUI 支持，Electron 内置了原生应用程序接口，对调用一些系统功能，如调用系统通知、打开系统文件夹提供支持。

相比较原生 C++等原生开发框架、QT 等跨平台框架来说，Web 技术跨平台桌面应用开发带来的是更加丰富的组件、更加灵活的技术架构、更快的开发周期以及更加繁荣的生态环境。但是其相比较原生，其系统性能和打包体积也是其主要瓶颈，也是本文想通过 WebAssembly 技术来进一步缩小弥补的性能差距。

2.2 高性能编码 WebAssembly 的研究

WebAssembly(简称 Wasm)是一个可移植、体积小、加载快并且兼容 Web 的全新编码格式，可以在现代网络浏览器中直接运行。它是一种低级的类汇编语言，具有紧凑的字节码格式，可以接近原生的性能运行。不仅能够保证代码执行的安全性，更重要的是为 Web 提供了解决底层代码执行速度，并且为其他的低等编译型语言提供一个编译目标，使得不同的代码都可以运行在 Web 端，并且提升了执行性能。它也被设计为可以与 JavaScript 共存，允许两者一起工作。

体积小：由于运行时只加载编译成的字节码，一样的逻辑比使用更高级解释型语言文件体积小很多。

加载快：由于文件体积小，再加上无需解释执行，WebAssembly 能更快的加载并实例化，并减少运行前的等待时间。

兼容问题少：WebAssembly 是非常底层的字节码规范，制定好以后很少变动，就算发生变化，只需要从高级语言编译成字节码过程做兼容。

目前 WebAssembly 仍然处于发展阶段，但针对 WebAssembly 的研究和应用一直处于被广泛关注的状态。WebAssembly 适合用于需要大量计算的场景，例如以下场景。在浏览器中处理音视频，flv.js 用 Web Assembly 重写后性能会有很大提升；React 的 dom diff 中涉及大量计算，用 WebAssembly 重写 React 核心模块能提升性能；Safari 浏览器使用的 JS 引擎 JavaScriptCore 也已经支持 WebAssembly。

2.3 对于网络代理技术的研究

网络代理相关的底层工具有很多，比如 WireShark、Whistle、Charles 等等，但很多互联网企业都基于底层架构做二次开发，以满足特定的差异化需求。

2.3.1 基于网卡的数据包捕获

网卡具有以下几种工作模式：

⑴ 广播模式:该模式下的网卡能够接收到网络中的广播信息。

⑵ 组播模式:该模式下的网卡能够接收组播数据。

⑶ 直接模式:工作在直接模式下的网卡只能接收目的地址是自己 Mac 地址的帧。

⑷ 混杂模式:工作在混杂模式下的网卡接收所有的流过网卡的帧,数据包捕获程序就是在这种模式下运行的。

因此,为了捕获到所有经过该网卡的数据包就必须先将网卡设置成混杂模式。这样,就可以捕获到流经该网卡的所有数据包。网卡主要的工作是完成对于总线当前状态的探测，确定是否进行数据的传送，判断每个物理数据帧目的地是否为本站地址，如果不匹配，则将它丢弃。如果是的话，接收该数据帧，进行物理数据帧的 CRC 校验，然后将数据帧提交给 LLC 子层。

2.3.2 基于代理的数据包捕获

通过在计算机上启动一个代理服务，并将计算机上所有请求用该服务代理，即所有请求都会优先经过该代理服务，然后再由代理服务分发到目标服务；服务返回结果也是先经过该代理服务，然后再由代理服务发送至计算机上的实际调用方。本小节重点叙述下 NodeJs 作为代理服务的好处和优势。

首先 NodeJs 采用事件驱动、异步编程，为网络服务而设计，有较高的吞吐量，适合 IO 密集型应用。而且非阻塞的 IO 处理也给 NodeJs 带来了在相对低系统资源消耗下的高性能与出众的负载能力，其轻量高效的特点，让其成为数据密集型分布式部署环境下的实时应用系统的完美解决方案。而网络代理服务恰恰符合这些特性，因为计算机上网络连接数量很多，同一时刻可能有多个端口都在与其他服务联结。通过代理服务，可以修改最终实际请求的报文信息，通过规则配置可以修改、拦截请求的头信息，也可以通过拦截返回并修改后，再返回给实际的请求方。

三、研究内容以及研究目标

3.1 研究目标

本毕业设计将基于 Node Proxy、Electron 以及 WebAssembly 架构技术作为基础，针对跨平台桌面网络代理系统做设计与实现，主要分为以下三个目标：

第一目标，功能目标，实现对网络数据包的捕获、对 HTTP/HTTPS 协议请求的代理、数据包解析、应用接口的迭代管理等主要功能模块。帮助开发人员快速切换依赖开发环境、接口数据 Mock 以及应用系统的生命周期管理。

第二目标，跨平台目标，针对不同的操作系统，可以正常使用系统功能。同时针对不同的操作系统底层接口和功能做差异化的编码方案，统一不同操作系统的使用体验。

第三目标，性能优化目标，针对 CPU 密集型计算，尤其是可视化部分、网络请求捕获分析模块的实现如何通过结合 WebAssembly 字节码来降低 JavaScript 解释型语言固有的性能瓶颈。

3.2 研究内容

主要的研究内容分为以下几个方面，

⑴ 首先是网络代理系统的需求分析，通过需求分析明确网络代理系统的基本功能和使用场景，明确开发过程中需要用到的技术。

⑵ 网络代理系统的设计，根据需求分析进行设计。主要是根据所掌握的技术，针对不同的技术场景合理规划。尤其是性能部分，使用相应的优化手段进行设计。

⑶ 网络代理系统的实现，根据设计的架构图、功能需求图、性能优化图等设计，进一步具体实现该系统。

⑷ 网络代理系统的测试，对实现的系统进行功能性测试和非功能性测试，同时对测试结果进行评估。

3.2.1 需求分析

首先，明确系统主要使用目标人群以及使用场景，主要是针对程序开发人员或者有一定网络代理需求的用户开发的一个工具类桌面系统应用，这个系统要满足跨平台、高性能、网络信息流捕获的要求，再根据捕获的数据进行软件生命周期管理的目标，基于此目标，我们将需求分为功能性需求和非功能性需求。

1、功能需求

分为四个主要方面网络请求代理、接口管理、包信息解析、接口生命周期管理。

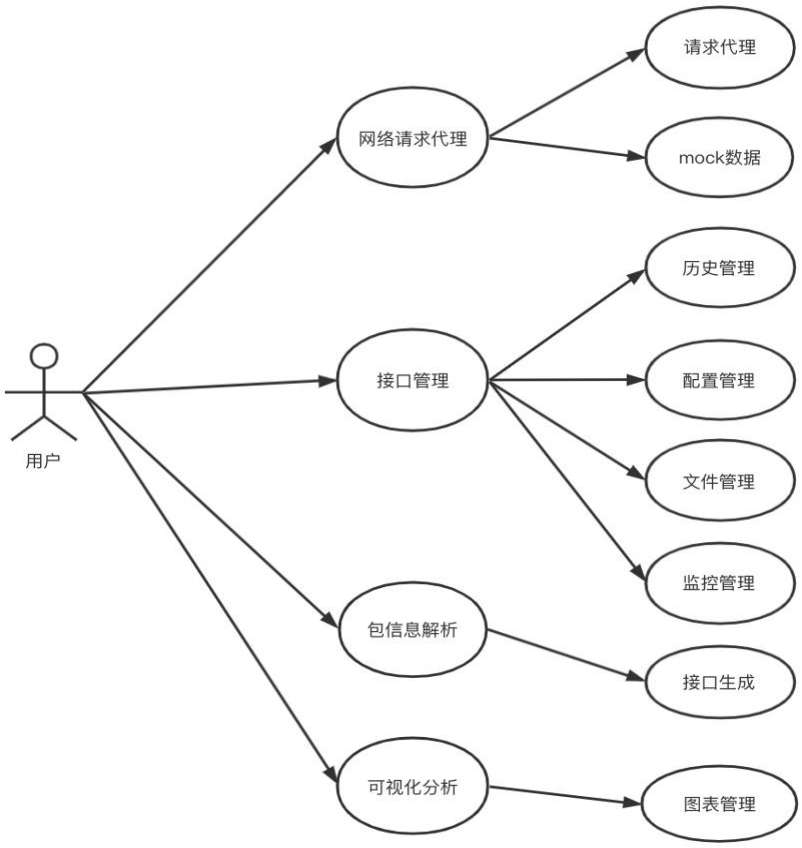

图 1 系统用例图

如图 1 所示是整个系统的用例图，可以拆分为四个模块

(1) 用例名称：网络请求代理用例

参与者：用户

说明：用户通过规则设置对特定请求进行请求头、mock 数据代理

前置条件：用户计算机系统成功设置代理服务

后置条件：请求按照我们的规则生效

(2) 用例名称：接口管理用例

参与者：用户

说明：用户通过设置请求的相关信息，成功在系统中请求，并对请求进行管理

前置条件：用户计算机系统成功设置代理服务

后置条件：请求正常工作并分类管理

(3) 用例名称：包信息解析用例

参与者：用户

说明：用户通过对捕获的网络请求包进行分析，解析并生成请求代码

前置条件：用户计算机系统成功设置代理服务并且对用户请求成功捕获

后置条件：请求按照我们的规则生效

(4) 用例名称：应用生命周期管理用例

参与者：用户

说明：用户在一段时间内所设置的特定接口进行数据管理、分析并生成相关的接口依赖、接口迭代、接口时间信息，丰富应用生命周期信息。

前置条件：用户计算机系统成功设置代理服务并且对用户请求成功捕获

后置条件：数据以时间为主线，多维度展开

2、非功能需求

(1) 跨平台

指可以在常见的不同种类的操作系统上，通过一套代码编译后的程序，都可以满足基本的使用。我们主要适配 Windows 操作系统和 MacOS 操作系统为主，因为这两类操作系统占有率已经高达 90%以上。

(2) 高性能

高性能主要是对比 C++/Rust 编译为 WebAssembly 相比传统的 JavaScript 的性能提升，以及 Chromium V8 引擎对代码运行时效率的优化、WebGL 引擎带来的渲

染性能提升。

(3) 安全性

遵循桌面应用开发道德底线，不破坏侵害目标操作系统，同时对自身安全、内存保证隔离安全。

(4) 可控性

作为桌面应用，要具备易恢复、重启动、可强制杀死进程等等

3.2.2 系统设计

系统设计的主要工作以需求为引导、根据实际情况对系统整体进行架构分层设计，定义层与层之间的交互，各个模块之间的协同性以及我们所要采用的技术选型和架构选型。系统设计的目的是帮我们从整体了解系统的结构，保障得到一个可以使用、满足需求、技术可行、稳定可靠的系统框架。为下一步系统实现打下坚实的理论依据。

1、系统架构设计

对系统整体架构进行分层设计，明确模块之间的依赖关系和交互逻辑。主要是针对不同的操作系统抽象出来的接口，通过 Electron 进行管理调用，结合 Electron 框架自身，使用 Proxy、React 等技术方案对特定模块做进一步的实现。

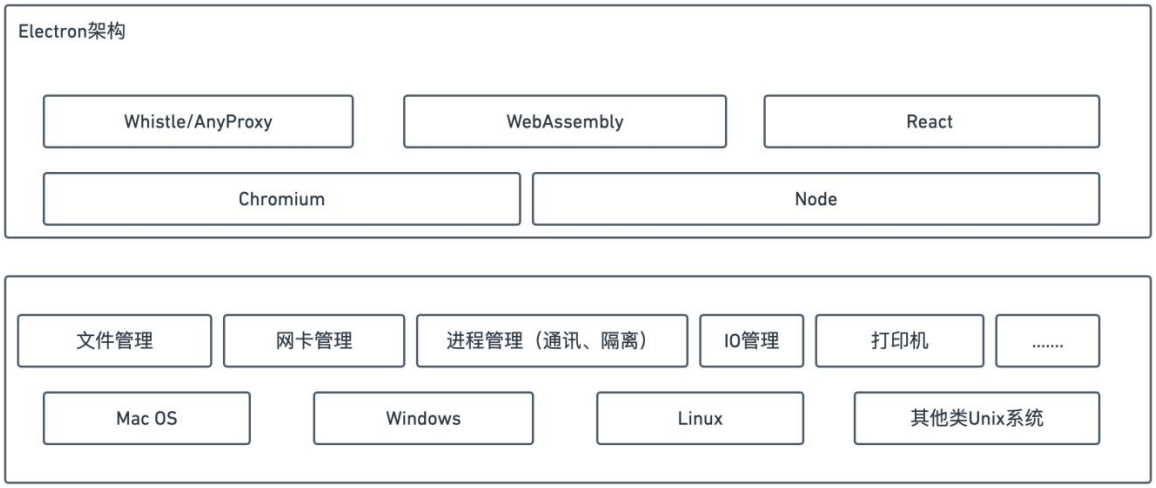

图 2 系统架构图

2、功能模块设计

以需求分析中功能需求为引导，设计系统整体的功能模块图，如图 3 所示：

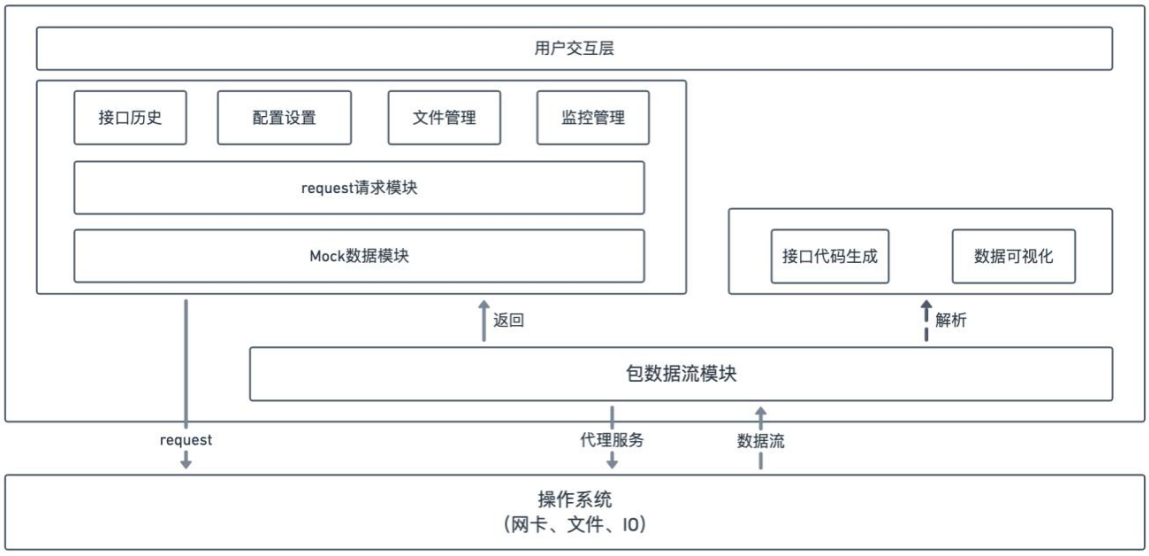

图 3 功能模块

系统功能模块主要核心是包数据流代理模块，基于请求的改造、请求的管理、历史信息的获取都是基于包数据流代理模块对信息的捕获。通过捕获到的元数据，进一步交给上层的接口管理模块和可视化信息模块做进一步的信息聚合、分析、展示。

其中最核心最根本的代理功能的设计方案，主要是采用代理服务的方式来实现对网络请求的捕获，其实现流程图六所示：

在实际调用方和服务方中间，我们通过代理服务来代理调用方的请求以及服务方的返回值，并通过自定义的分析处理模块可以对包信息进行任何处理，从而实现我们的功能。

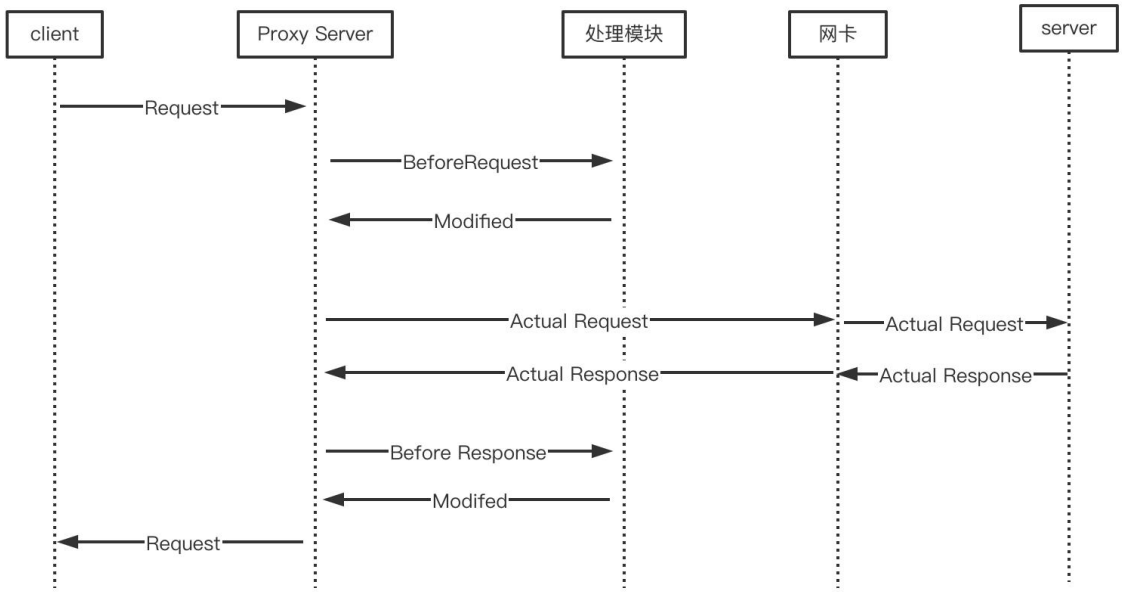

图 4 代理过程图

3、性能需求设计

(1) 用编译型语言代替解释型语言

编译型语言比解释性语言执行效率更高，宏观角度在于编译型语言更加接近底层、通过提前编译提高了运行时的速度。通常把 C++、Rust 称作编译型语言，而 JavaScript、Python 等叫做解释型语言。如图六所示，是编译型语言和解释型语言通用的执行流程。

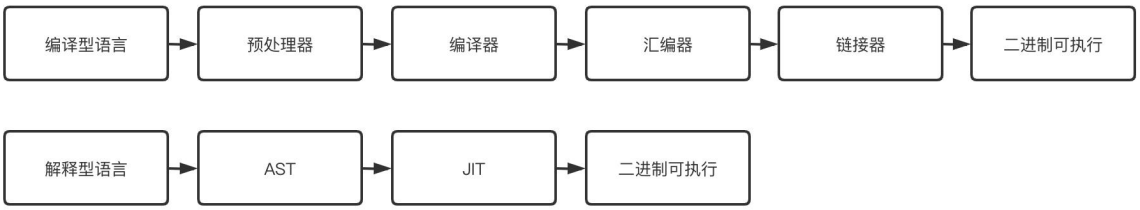

图 5 编译型语言和解释性语言

如图 6 所示，可以理解 WASM 已经是比较接近机器语言的二进制编码,相比较 JavaScript 的执行流程，它可以尽快的在运行时被编译为机器码执行，而 JavaScript 要经过 V8 的 pipeline，然后在被编译。相比较之下，WASM 代码的执行速度会更快。

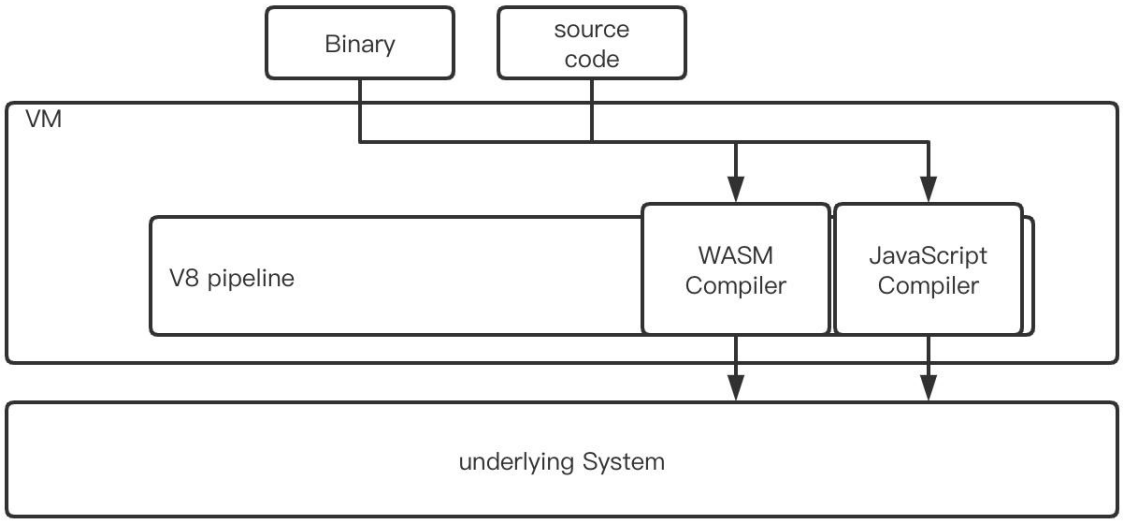

图 6 WASM 和 JavaScript 代码在 V8 引擎中的加载对比

(2) 编码方案

二进制编码在 V8 运行时的加载流程时机。

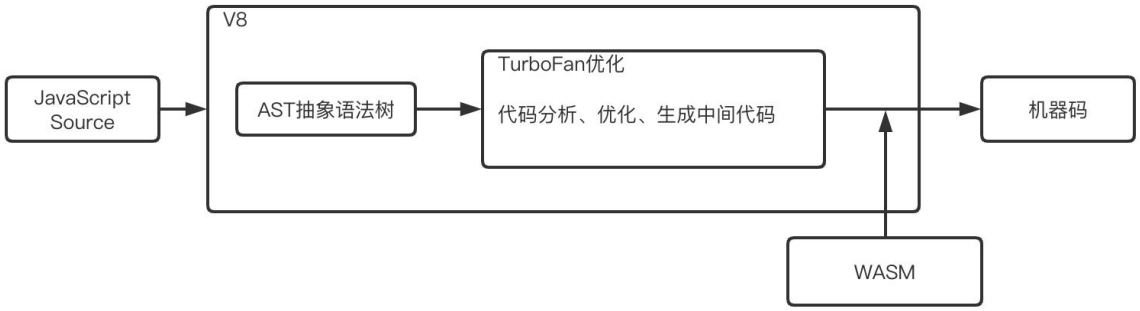

图 7 WASM 的加载时机

WASM 采用了一种基于小端模式的编码算法进行可变长编码。WASM 对不同的数据类型选择了不同的编码方案，最大限度的利用各个编码方案的优点。

1、部分整数类型数值编码

基于 LEB-128 的整数编码，是一种用于整数的、基于小端模式的可变长编码，所以可变长编码是指待编码的源数据在经过编码算法后得到的编码结果长度是不固定的。通过使用可变长对源数据进行无损数据压缩，并且被压缩后的数据也可以随时被再次解压缩回源数据，通过合理编码压缩，保证了编译后的模块体积处于最优。

2、浮点数编码 IEEE-754 编码

3、字符串编码 UTF-8 字符编码

(3) 自定义虚拟指令集

虚拟指令集是用于构成 WASM 模块核心功能的关键元素之一。需要让执行环境理解我们编码的意思，必须要让执行环境先理解指令集，然后根据指令集再对二进制编码进行编译，进而被执行。

3.2.3 系统实现

完成系统的设计后，利用 JavaScript 代替原来的 GUI 开发桌面应用。对于

CPU 密集型计算性能消耗，比如大量的数据包捕获和解析、渲染图像，通过使用 C++或 Rust 编程语言编译成自定义字节码和 WebGL 渲染引擎来优化来进行开发。最后根据系统架构设计和功能模块设计，将业务需求和功能模块串联起来，实现网络代理系统的功能。

3.2.4 系统测试

在完成系统开发后，我们进行系统的功能性测试和非功能性测试。

对于功能测试，需要在不同场景下验证系统的功能，主要使用灰盒测试、自动化测试对系统中的模块进行联合测试和单独测试。我们采用一个复杂的线上开发应用的 API 作为测试目标，查看是否可以对 API 进行捕获、环境切换、数据包 Mock 等。

对于非功能测试，系统需要满足跨平台必须满足在 Window、Mac、Linux 实现可下载安装、可正常使用大部分功能。高性能需要对比同等条件下，与不使用 WASM 编码方式做横向比较。

四、实施方案与可行性分析

4.1 研究基础

桌面跨平台技术是追求开发一致性以及抹平底层差异，解放程序员生产力的重要技术。WASM 编码技术让缩小了 JavaScript 语言本身带来的性能问题，WebGL 渲染引擎提供了高效的图像渲染能力。网络代理工具更是保障程序员在多线上环境、本地开发的情况下，提升开发体验，提高效率，保障应用的生命周期。本论文要实现的桌面应用系统的功能需求和非功能需求，功能需求包括：网络请求代理、接口管理、包信息解析、应用生命周期管理；非功能需求主要包括：实现跨平台和高性能编码和 WebGL 提高性能满足大量网络请求捕获和可视化图像渲染。

4.2 主要技术方案

4.2.1 Electron 框架

Electron 结合了基于 V8 引擎的轻量浏览器内核 Chromium 和 NodeJs 丰富强大的系统层面的接口，高效利用了操作系统的能力，使得可以通过 JavaScript 来创建跨平台的桌面应用。

1、 NodeJs

NodeJs 是一个 JavaScript 运行环境，是对 Google V8 引擎进行了封装，用于方便地搭建响应速度快、易于扩展的网络应用。Node.js 使用事件驱动、非阻塞 I/O 模型而得以轻量和高效，非常适合在分布式环境中运行数据密集时实时应用。

2、系统 API

为了提供原生系统的 GUI 支持，Electron 内置了原生应用程序接口，对调用一些系统功能，如调用系统通知、打开系统文件夹、访问操作系统剪贴板等提供支持。

3、进程类别

Electron 区分了两种进程：主进程和渲染进程。

一个 Electron 应用总是有且只有一个主进程，主进程职责：

⑴ 创建渲染进程（可多个）；

⑵ 控制应用生命周期（启动、退出 APP 以及对 APP 做一些事件监听）；

⑶ 调用系统底层功能，调用原生资源。

一个渲染进程相当于一个桌面应用窗口，其主要职责：

⑴ 用 HTML 和 CSS 渲染界面；

⑵ 用 JavaScript 做一些界面交互。

Electron 技术优势，桌面应用利用了 Web 技术的丰富性并通过其他 V8 引擎等保障运行时的效率和性能。可以即时启动，不需要等待资源从网络下载下来。可

以访问计算机的操作系统和硬件资源，包括可以读写用户计算机中的文件系统。可以更好地控制软件的用户体验，不需要担心兼容性问题。可以用 Web 前端技术开发跨平台的桌面应用：使用纯 JavaScript 语言开发，只需要写一份代码，打包出来的应用可以同时在 Windows、Linux、Mac 操作系统上运行。可以从 Node.js 的生态获得极大的助力：Node.js 这个大生态下很多成熟模块可以直接引入使用，避免重复造轮子，提高开发效率。进程隔离：基于 Chromium 多进程模式的应用模块集成，天然提供了应用模块之间的隔离性，其中某一应用模块的故障不影响其他应用模块及整个应用软件。

4.2.2 WASM

WebAssembly 是可移植、体积小、加载快并且兼容 Web 的一种新的字节码格式，主流浏览器都已经支持 WebAssembly。WebAssembly 字节码和底层机器码很相似，可快速装载运行，因此性能相对于 JavaScript 解释执行大大提升。

对于网络平台而言，WebAssembly 具有巨大的意义——它提供了一条途径，使得以各种语言编写的代码都可以以接近原生的速度在 Web 中运行。以下主要阐述基于 WebAssembly 的计算架构的技术特性。

跨平台性

WebAssembly 字节码是一种抹平了不同 CPU(central processing unit，中央处理器）架构的机器码，WebAssembly 字节码不能直接在任何一种 CPU 架构上运行，但由于非常接近机器码，可以非常快地被翻译为对应架构的机器码，因此 WebAssembly 运行速度和机器码接近。每个高级语言源码到不同平台的机器码的转换工作是重复的，高级语言只需要生成底层虚拟机（low level virtual machine,LLVM）认识的中间语言（LLVM intermediate representation,LLVM IR),LLVM 能实现 LLVM IR 到不同 CPU 架构机器码的生成、机器码编译时性能和大小的优化。除此之外，LLVM 还能实现 LLVM IR 到 Web Assembly 字节码的编译功能，也就是说只要高级语言能转换成 LLVM IR，就能被编译成 Web Assembly 字节码。

4.3 可行性分析

自身掌握开发中所需要的的编程语言 JavaScript 和 C++，对跨平台设计有一定了解，前期通过阅读文献和在公司实际开发实习积累相关的经验，熟悉了在开发过程中使用的 React、WebAssembly、Electron 等等一些基本框架和技术，初步分析表明本方案具备可行性。校内外导师对相关领域具备专业能力，对论文撰写具有指导学生的经验。

4.4 学习计划

时间	内容
现在 ~ 4.15	完成基础框架的搭建测试，主要是对网络代理功能的实现
4.16 ~ 5.15	深入学习 WebAssembly 的编码与运用
5.16 ~ 7.15	完成接口生命周期管理的模块、数据包反解析分析以及性能需求
7.16 ~ 9.15	完成系统性能部分的开发、测试以及开始论文撰写
9.15 ~	完成论文撰写以及后续工作

五、参考文献

[1]罗青林,徐克付,臧文羽,刘金刚.Wireshark 环境下的网络协议解析与验证方法[J].计算机工程与设计,2011,32(03):770-773.DOI:10.16208/j.issn1000-7024.2011.03.068.

[2]ROSSBERG, ANDREAS, TITZER, BEN L., HAAS, ANDREAS, et al. Bringing the Web Up to Speed with WebAssembly[J]. Communications of the ACM,2018,61(12):107-115. DOI:10.1145/3282510.

[3]RAY VILLALOBOS. NW.JS VS ELECTRON[J]. Net,2016(Sep. TN.284):111.

[4]DONGSEOK JANG, RANJIT JHALA, SORIN LERNER, et al. An Empirical Study of Privacy-Violating Information Flows in JavaScript Web Applications[C]. //17th ACM Conference on Computer and Communications Security.:Curran Associates, Inc., 2010:270-283.

[5]KYRIAKOU, KYRIAKOS-LOANNIS D., TSELIKAS, NIKOLAOS D., KAPITSAKI, GEORGIA M.. Enhancing C/C plus plus based OSS development and discoverability with CBRJS: A Rust/Nodejs/WebAssembly framework for repackaging legacy codebases[J]. The Journal of Systems and Software,2019,157. DOI:10.1016/j.jss.2019.110395.

[6]SANTIAGO ESCOBAR. Mechanising and verifying the WebAssembly specification[J]. Computing reviews,2019,60(1):24.

[7]ROSSBERG, ANDREAS, TITZER, BEN L., HAAS, ANDREAS, et al. Bringing the Web Up to Speed with WebAssembly[J]. Communications of the ACM,2018,61(12):107-115. DOI:10.1145/3282510.

[8]Haas A, Rossberg A, Schuff D L, et al. Bringing the web up to speed with WebAssembly[C] //ACM SIGPLAN Notices. ACM, 2017, 52(6): 185-200.

[9]Wonsun Ahn,Jiho Choi,Thomas Shull,María J. Garzarán,Josep Torrellas. Improving JavaScript performance by deconstructing the type system[P]. Programming Language Design and Implementation,2014.

[10]Toman Zinah Hussein,Toman Sarah Hussein,Hazar Manar Joundy. An In-Depth Comparison Of Software Frameworks For Developing Desktop Applications Using Web Technologies[J]. Journal of Southwest Jiaotong University,2019,54(4).

[11]Conrad Watt. Mechanising and verifying the WebAssembly specification[P]. Certified Programs and Proofs,2018.

[12]Maria Cristina ENACHE. Cross-Platform Technologies[J]. Annals of Dunarea de Jos University. Fascicle I : Economics and Applied Informatics,2017,23(1).

[13]Adam Rapley,Xavier Bellekens,Lynsay A. Shepherd,Colin McLean. Mayall: A Framework for Desktop JavaScript Auditing and Post-Exploitation Analysis[J]. Informatics,2018,5(4).

[14] Kredpattanakul, Kitti & Limpiyakorn, Yachai. (2019). Transforming JavaScript-Based Web Application to Cross-Platform Desktop with Electron: ICISA 2018. 10.1007/978-981-13-1056-0_56.

[15]薛超. 基于 WebAssembly 的 JavaScript 性能优化方案研究与实现[D].西北大学,2019.

[16]万里晴,杨浩.探究基于 V8 引擎的 Node.js 在各应用领域的发展[J].通讯世界,2015(13):97.

[17]河南大学. 一种基于 electron 的跨平台桌面应用程序开发框架及方法:CN201910520250.0[P]. 2019-09-24.

[18]胡佳静. 基于 electron 的待办事项管理 app 开发[D]. 湖北:华中科技大学,2018.

[19]褚孔统,朱勇.开发跨平台桌面应用的探讨[J].机电信息,2019(33):55-56.DOI:10.19514/j.cnki.cn32-1628/tm.2019.33.030.

[20] 朱丽英. 基于 Node-Webkit 平台的 JavaScript 工具集研究与实现[D]. 2016. DOI:10.7666/d.D00988796.

[21] Paul B.Jensen. 跨平台桌面应用开发基于 Electron 与 NW. JS.北京: 电子工业出版社, 2018

[22]邓杰海.应用 Electron 架构技术操作 Excel 文件[J].现代计算机,2019(32):91-94.