编 号

# 本科生毕业设计（论文）

题目：新一代无插件交互式 Web 技术研究及应用

数字媒体 学 院 数字媒体技术 专 业

学　　号 0305100309

学生姓名 谭林权

指导教师 夏鸿斌　副教授

二〇一四年六月

# 摘 要

近几年来，现代互联网技术高速发展，并且越来越来朝多样化的方向发展。加上现在的人的时间观念，传统的Web通信技术已经不能满足人们的需要。并且，多平台多终端多版本多浏览器的复杂问题，也导致了像Flash一类使用插件的手段已经不能很好的满足人们的需求，不能带来完美的交互体验。

基于浏览器，开发人员可以利用HTML、CSS、JavaScript来开发Web应用。HTML5出来过后，开发人员更是能够创造令人瞠目结舌的应用，这些应用足以媲美相对体积庞大客户端应用，给人们带来前所未有的交互体验。

本文主要是对HTML5的Canvas和WebSocket技术进行详细深入的研究，其中包括对Canvas标签api的详细解读与试验，对WebSocket全双工实时通信技术的前后端接口实现，Web实时通信的详尽流程实现进行了研究与试验。并且最后利用这些技术，结合自己的构思及设想，以及使用node.js、socket.io作为服务端支撑及浏览器实时通信整合，开发了一款Web实时交互的应用示例。

本文的研究过程以及研究成果，对如何开发新一代无插件交互式Web应用，特别是实时交互式应用的架构和开发具有一定的指导意义。

**关键词：**Web；实时；交互；HTML5；WebSocket

# ABSTRACT

In recent years, the modern Internet technology developed with high speed, and increasingly to toward the direction of diversification.Now,combined with people’s concept of time, the traditional web communication technology already cannot satisfy people's needs.And multiple platform terminal more version of the browser's complex problems, also contributed to the Flash using plug-in means has already can not meet people's needs well, bring the terrible interaction experience.

Based on the browser, developers can only use HTML, CSS, and JavaScript, to develop the web applicationsWhen HTML5 comed, developers are able to create stunning applications, these applications as well as relatively large client application, it brings unprecedented interaction experience.

This article is mainly to the HTML 5 Canvas and WebSocket technology to research in detail, including a detailed interpretation of tag API with the experiment, the WebSocket full-duplex end interface before and after the implementation of real-time communication technology, real time web communication of detailed process implementation are researched. And of these technologies, combined with my own idea and view, and use the Node.js, socket. io as server browser support and real-time communication integration librabry, finally developed a real-time interactive web application as a example.

In this article, the research process and research achievements, for how to develop a new generation of no plug interactive web applications, in particular, real time interactive applications architecture and development has a certain guiding significance.

**Keywords**: Web;Real time;Interactive;Html5;WebSocket

# 目 录

# 第 1 章 绪论

## 1.1 选题依据及意义

### 1.1.1 研究背景

二十世纪九十年代初，万维网的发明者蒂姆•伯纳斯-李(Tim Berners-Lee)在 alt.hypertext 讨论组上贴出了一份关于其研究项目万维网的帖子。这个帖子说明这个项目是为了构建链接系统，即“允许建立指向任意地点任意信息的链接”。万维网从那天开始名副其实。

伯纳斯-李多年来一直在欧洲粒子物理研究所物理实验室从事超文本信息服务的研究。在 1990 年末，他就完成了第一个万维网服务器及浏览器的编写，但是直到 1991 年 8 月 6 日，他才将其对外公布。

伯纳斯-李将这些程序放在 HTTP://info.cern.ch/的服务器上，并在该网站上发布了一些关于该项目的信息，这便是世界上第一个网站。不到几个星期的时间，人们便从服务器上下载这些代码，并以此创建属于他们自己的网站。如此，网络就这样被编织了起来。尽管 8 月 6 日只是一个人为的纪念日，但这一天的确值得用来纪念，因为就是在这一天，Web 走向了世界,和许多永远只能沉寂在实验室里的研究项目作比，互联网的发明改变了整个世界。

1996 年 11 月，美国 Macromedia 公司（后被 Adobe 公司收购）收购了 Future Wave，Flash 正式诞生。Flash Player 能够播放多媒体动画以及交互式的动画，而且也支持文字输入、高品质的 Mp3 音频流、友好的交互式接口等。这个播放器的体积一点儿都不大，下载只需要花一点点时间，用来体验发布在网页上的多媒体效果是个绝佳的开始。因此，那个年代，如果你想要在网页上观看多媒体内容，Flash Player 几乎是一个必装插件，为它所制作的动画或图像随处可见。

### 1.1.2 问题的提出

二十一世纪第一个十年过去，移动互联网几乎是爆炸式的进入人们的视野，移动互联网的时代正式到来。新的移动平台，给 Web 带来了更多的机会，但同时也面临着更多的挑战，原来 PC 端的各种浏览器兼容，转眼就已经升级为多平台多终端多版本多浏览器的复杂问题。这个时代，像 Flash 这种需要插件支持的形式，已经不能满足跨平台，跨多终端的需求了。于是，HTML5 标准应运而生，Canvas、WebSocket 等新的 API 恰如其分的从很大程度上解决了这个棘手的问题。使用 HTML5 开发的新一代 Web 应用程序，带来了前所未有的 Web 交互体验，因此，许多知名研究人员预言，HTML5 将是新一代无插件交互式 Web 技术的焦点。为什么实时 Web 这么重要？因为我们生活在一个实时 (real-time)的世界中，因此 Web 的最终最自然的状态也应当是实时的。

## 1.2 国内外研究现状

由于 HTML5 标准比较前沿，所以导致一些比较老的浏览器不能够很好的支持这些新的属性，这些浏览器主要是 IE 系列。在国内，包括 IE6 在内的用户数量还占有很大的比例。所以，为了顾及每一个用户都能正常访问站点，很多公司退而求其次，放弃了使用新的技术。但是，随着浏览器的更新换代，会有越来越多的公司利用 HTML5 做出令人瞠目结舌

的交互作品。而在国外，思想比较开放，技术同样走得比较前沿，所以已经有很多公司开始全面使用 HTML5。但是无论在国内还是国外，Web 实时交互却一直以来是一个比较新的领域，期待着更多的人去探索，去发现。

## 1.3 研究思路及主要内容

在新的 HTML5 标准中，Canvas 元素用于在网页上绘制图形，该标签强大之处在于可以直接在 DOM 上进行图形操作，具有极大的应用价值。Canvas 标签本身其实并不具有绘图的功能，而是通过借助 JavaScript 实现在网页上绘制图像。Canvas 画布定义了一个矩形区域，其中的每一个像素我们都可以控制。Canvas 拥有多种绘制路径、圆形、矩形、字符和图像的接口，可以用来开发丰富的图形应用。并且，开发完成过后，用户不需要下载任何插件，便能够在多平台、多终端的现代浏览器中使用。

WebSocket 是 HTML5 标准中目前最强大的通信功能，它定义了一个全双工的通信信道，仅通过 Web 上的一个 Socket 即可实现通信。WebSocket 不仅仅是对常规 HTTP 通信的一次增量升级，它更代表着一次巨大的进步，对实时、事件驱动的 Web 应用程序来说更是如此。目前实时 Web 应用的解决方案，基本上是围绕轮询以及其他服务器端推送技术，比如 Comet 和 Ajax 轮询（polling）、长轮询(long-polling)、流（streaming）、Flash 等，所有这些提供实时数据的方案都包含有大量额外不必要的报头数据，造成传输延迟。最重要的是为了在半双工 HTTP 的基础上来模拟全双工通信，目前大部分解决方案都是使用了两个连接：一个用于上行数据流，另一个用于下行数据流。这两个连接的保持和协作会造成大量的资源消耗，并增加了复杂度。建立一个 WebSocket 通信，需将 HTTP 协议升级到 WebSocket 协议，在客户端和服务器在初次握手的时候。

本科题也将借助 html5 的 canvas 和 WebSocket，以及 nodejs 等技术，对 web 实时交互技术进行系统深入的研究，并在最后做出一个无插件 Web 实时交互的应用。

# 第 2 章 开发环境介绍

## 2.1 简介 Linux Ubuntu 操作系统

### 2.1.1 Linux 系统

Linux 操作系统是一套免费使用而且能够自由去传播的类 Unix 操作系统，是基于 POSIX 和 UNIX 的多用户、多任务、支持多线程和多 CPU 的一个操作系统。Linux 操作系统能够运行主要的 UNIX 工具软件、应用程序以及网络协议。Linux 操作系统支持 32 位硬件和 64 位硬件。Linux 操作系统继承了 Unix 以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。


图 2-1 Linux 操作系统标志

Linux 操作系统的诞生日期大家统一规定为 1991 年 10 月 5 日（这是 Linux 第一次正式向外公布的时间）。Linux 存在着各种不同的 Linux 版本，但它们有一个共同点，那就是都使用了 Linux 内核。Linux 可以被安装在各种计算机硬件设备里面，比如手机、平板电脑、路由器、视频游戏控制台、台式计算机、大型机，以及超级计算机。从某种严格意义上来讲，Linux 这个词本身只代表了 Linux 内核，但现实工作生活中，人们早就已经习惯了用 Linux 来形容整个基于 Linux 内核，并且使用 GNU 工程各种工具和数据库的操作系统。

### 2.1.1 Ubuntu 系统

Ubuntu（乌班图）是一个 Linux 操作系统，其特点是以桌面应用为主，其名称来自非洲南部祖鲁语或豪萨语的“ubuntu”这个词语，它的意思大概是“人性”、“我的存在是因为大家的存在”，是非洲传统的一种价值观的体现，就像华人社会的“仁爱”思想一样。Ubuntu 操作系统是基于 Debian 发行版和 GNOME 桌面环境，它每 6 个月就会发布一个新版本，这是与 Debian 不同的地方。

图 2-2 Ubuntu 发行版标志

Ubuntu 操作系统的目标在于为普通用户提供一个最新的、并且又十分稳定的主要由自由软件构建而成的一个操作系统。Ubuntu 操作系统具有十分庞大的社区力量支持，因此，大家可以十分方便地从社区来寻求帮助。本课题研究使用的为 Ubuntu14.04 LTS 版本，代号为可信赖的塔尔羊。

## 2.2 简介 Node.js

### 2.2.1 谷歌 V8 引擎

V8是一个开源JavaScript引擎，由丹麦谷歌，最早用于Google Chrome浏览器中。目前该JavaScript引擎已用于其它项目的开发。第一个版本随着第一个版本的Chrome于2008年9月2日发布。

V8使用C++开发，并在谷歌浏览器中使用。在运行JavaScript之前，相比其它的JavaScript引擎转换成字节码或解释执行，V8却是将其编译成原生机器码（IA-32, x86-64, ARM, or MIPS CPUs），并且还使用了例如如内联缓存（inline caching）等方法来提高性能。因为实现了这些，JavaScript程序在V8引擎下的运行速度可以和二进制程序相媲美。

V8可以独立运行，也可以嵌入到任何C++应用程序。项目托管在Google Code上，基于BSD协议，任何组织或个人都可以将其源码在自己的项目中使用。

### 2.2.2 Node.js 简介

Node.js是一个可以用来快速构建应用及网络服务的平台。这个平台是基于Chrome's JavaScript runtime来构建的，从另一个角度来说，它实际上就是是对Google的V8引擎进行了封装。

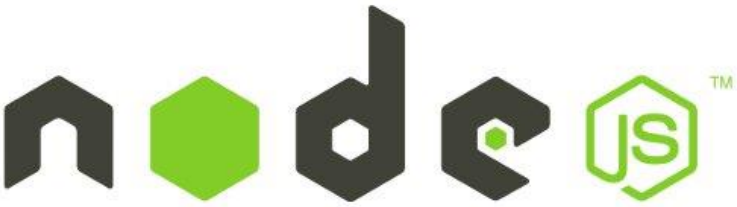


图 2-3 Node.js 标志

V8引擎执行JavaScript不仅速度非常快，而且性能也十分优秀。Node.js对一些特殊用例进行了优化，然后提供了替代的API（接口），这使得V8即时在非浏览器环境下，能够运行得更好。比如，在通常的服务器环境中，处理二进制数据一般来说是必不可少的，但

JavaScript却恰恰对此支持不足。因此，基于V8的Node.js专门增加了Buffer这个类，用来方便并且高效地处理二进制数据。因此，Node.js不仅仅简单地使用了V8引擎，还对其进行了一定程度的优化，使其在各种环境能够运行的更加出色。由于V8引擎本身使用了一些最新的编译技术，这使得用JavaScript这类脚本语言编写出来的代码运行速率获带来了非常大的提高，并且节省了开发成本。

JavaScript是一个事件驱动语言，Node.js很好的利用了JavaScript这个优点，使其可以编写出高扩展性的服务器脚本。Node.js采用了一个被称为“事件循环(event loop）”的架构，正因为这样，Node.js编写高扩展性的服务器变得不仅容易而且安全性能也很高。提高服务器性能的技巧有着成千上万的技巧，Node.js选择了一种既能提高性能，同时又能减低开发复杂度的架构，这是Node.js的一个异常重要的特性。并发编程从某种意义上说通常不仅十分复杂而且还满地地雷。Node.js恰恰另辟蹊径绕过了这些，而且仍能够提供优秀的性能。Node.js采用一系列“非阻塞”库来支持事件循环(even loop)。本质上来说就是为文件系统、数据库之类的资源提供了一系列的接口。例如我们向文件系统发送一个请求时，无需等待硬盘（寻址并检索文件），硬盘准备好的时候非阻塞接口会自然会知道通知Node.js。该模型以可扩展的方式简化了对慢资源的访问， 不仅直观，而且易懂，尤其是对于熟悉DOM事件的开发者来说，更有一种似曾相识的感受。

虽然让JavaScript能够运行在服务器端不是Node.js一家所能够做到的，但却是Node.js其一强大的功能。浏览器提供的宿主环境限制了开发者选择编程语言的自由度。任何服务器和越来越复杂的浏览器客户端应用程序间共享代码的方式只能通过JavaScript来实现。虽然还存在其他一些支持JavaScript运行在服务器端的平台，但因为上述特性，Node.js发展速度十分迅猛，已经成为一个成为事实上的平台。在Node.js项目启动的很短时间内，社区就已经贡献了大量的扩展库（模块）。其中大多是连接数据库或是其他软件的驱动，但凭他们的实力制作出来的非常有用的软件也是数目不小的。

虽然Node.js项目诞生才短短一段时间，但很少看到对一个项目如此狂热的社区。不管是新手，还是技术专家，大家都围绕着项目来使用并且贡献自己的一份力量，致力于打造一个探索、支持、分享、听取建议的技术社区。

### 2.2.3 Node.js 的优点

Node.js是一个新兴的后台语言，RESTful API、单线程，这是它十分吸引开发者的特点。

Node.js可以不新增额外线程，依然能够实现对多任务进行并行处理（其实Node.js是单线程的）。它通过事件轮询（event loop）来实现并行操作，对此，我们应该要充分利用这一点，尽可能的避免阻塞操作，取而代之，多使用非阻塞操作。

Node.js拥有十分优秀的性能。创始人Ryan Dahl的曾经说到，性能是Node.js项目考虑的重要因素，为什么选择C++和V8，而不是Ruby或者其他的虚拟机也是基于性能的目的。Node.js在设计上让人感觉相当胆大，它以单进程、单线程模式运行（和JavaScript的运行方式一致），事件驱动机制是Node.js通过内部单线程高效率地维护事件循环队列来实现的，

其中省去了多线程的资源占用和上下文切换，这意味着面对大规模的HTTP并发请求，Node.js凭借事件驱动似乎就能够解决问题。习惯了传统语言的网络服务开发人员可能对多线程并发和协作非常熟悉，但是面对Node.js这门新兴的语言，我们需要耐心的接受和理解它的特点。淘宝共享数据平台团队对Node.js的性能测试数据如表2-1所示。

表 2-1 淘宝共享数据平台团队对 Node.js 的性能测试

| | |
|---|---|
| 物理机配置 | RHEL 5.2、CPU 2.2GHz、内存 4G |
| Node.js 应用场景 | MemCache 代理，每次取 100 字节数据 |
| 连接池大小 | 50 |
| 并发用户数 | 100 |
| 测试结果（socket 模式） | 内存（30M）、QPS（16700）、CPU（95%） |

这个性能测试表说明，在这样的测试场景下，qps能够达到16700次，内存却仅仅占用30M（其中V8堆还占用了22M，也就是说数据仅占用了8M）。但我们也发现CPU的占用达到了95%，这可能会成为一个性能瓶颈。此外，还有不少实践者对Node.js做了性能分析，总的来说，它的性能让人信服。既然Node.js采用单进程、单线程模式，那么在如今多核硬件流行的环境中，单核性能出色的Node.js如何利用多核CPU呢？创始人Ryan Dahl给出了一个建议，我们运行多个Node.js进程，利用某些通信机制来协调各项任务，从而达到最佳处理效果。

## 2.3 HTML 、CSS 和 JavaScript

### 2.3.1 HTML

超级文本标记语言（Hyper Text Markup Language）。页面内可以包含图片、链接，甚至音乐、程序等非文字元素，这就是通常意义上的超文本。超级文本标记语言是标准通用标记语言分支下面的一个应用，同样它也是一种规范，一种标准，它通过标记符号来标记网页中需要显示的内容。网页文件本身是一种文本文件，通过在文本文件中添加标记符，可以告诉浏览器应该如何渲染显示这些内容（例如：文字怎么处理，画面怎么安排，图片怎么展现出来等）。浏览器默认是按顺序来阅读网页文件的，然后根据标记符解释和显示其标记的内容，但错误的地方浏览器是不会给我门指出来的，并且其解释过程也不会停止。网页文件书写者只能通过显示效果来分析出错原因和具体的出错部位。同样让人觉得不能理解的是，对于不同的浏览器，对同一标记符却可能会有不完全相同的解释，而且可能会造成与众不同的显示效果。

### 2.3.2 CSS

CSS，也就即级联样式表。它是一种用来表现上小结HTML或者XML（标准通用标记语言的一个子集）等文件样式的一种比较常见的计算机语言。

CSS3是目前CSS的最新版本，和2.0不同的是它新增了很多令人惊讶的特性，是能够真正做到将网页表现与内容分离的一种样式设计语言。相对于传统HTML的展示的结构式表现来说，CSS能够对网页中的对象的位置排版进行像素级的精确控制，支持几乎所有的字体字号样式，拥有对网页对象和模型样式编辑的能力，并且能够实现初步的网页交互，是目前基于文本展示领域最优秀的表现设计语言。CSS能够根据不同使用者的理解能力，简化或者优化写法，针对各类人群，而且也十分容易阅读。

### 2.3.3 JavaScript

JavaScript是一种由Netscape的LiveScript发展而来的基于原型的基于对象的动态类型的区分大小写的一种客户端脚本语言，主要设计目的是为了解决服务器端语言，遗留的速度问题，以便为客户提供更为流畅的浏览体验。当时服务端需要对数据进行验证，由于网络速度相当缓慢，只有28.8kbps，验证步骤浪费的时间太多。于是Netscape的浏览器Navigator默默的加入了JavaScript，为浏览器提供了数据验证的相关接口。

JavaScript是一种基于对象和事件驱动并且具有相对安全性的客户端脚本语言。同时也是一种广泛用于客户端Web开发的脚本语言，常用来给网页添加动态功能，比如响应用户的各种命令。它最初由网景公司（Netscape）的Brendan Eich设计，是一种动态、弱类型、基于原型的语言，内置支持类。JavaScript是Sun公司(已被oracle收购）的注册商标。Ecma国际以JavaScript为基础制定了ECMAScript标准。完整的JavaScript实现包含三个部分：ECMAScript，文档对象模型（DOM）和浏览器对象模型（BOM）。

JavaScript最初受Java启发而开始设计的，目的之一就是“看上去像Java”，因此语法上有类似之处，一些名称和命名规范也暗暗模仿Java。但JavaScript的主要设计原则源自Self和Scheme。JavaScript与Java名称上的近似，本质上是当时网景为了营销考虑，从而与Sun公司达成协议的结果。

ECMAScript-262是JavaScript标准，基于网景（Netscape）公司提出JavaScript语言和微软公司提出的JScript语言。ECMA开始于1996年，在1997年7月，采纳了首个版本，1998年，该标准成为了国际ISO标准.

布兰登•艾奇（Brendan Eich，1964年～），JavaScript的发明人，从2007年开始在Mozilla公司担任首席技术长官（Chief Technology Officer）。2014年3月25日，Mozilla宣布由原技术长兼联合创始人Brendan Eich接任新执行官职位，同时现任执行官Gary Kovacs正式退位。

## 2.4 Canvas

Canvas对象表示一个 HTML画布元素<canvas>。是为了客户端矢量图形而设计的，它没有自己的行为，但是定义了一个 API 支持脚本化客户端绘图操作。你可以直接在该对象上指定宽度和高度，但是，其大多数功能都可以通过CanvasRenderingContext2D 对象获得。这是通过 Canvas 对象的getContext()方法并且把直接量字符串 "2d" 作为唯一的参数传递给它而获得的。Canvas API使用了路径的表示法。但是，路径由一系列的方法调用来定义，而不是描述为字母和数字的字符串，比如调用 beginPath()和arc()方法。一旦定义了路径，其他的方法，如fill()，都是对此路径操作。绘图环境的各种属性，比如 fillStyle，说明了这些操作如何使用。

<canvas> 标记在Safari 1.3中引入，在 Firefox 1.5 和 Opera 9 中也得到了支持。我们甚至可以在 IE 中使用 <canvas> 标记，并在 IE 的 VML 支持的基础上用开源的 JavaScript 代码（由 Google 发起）来构建兼容性的画布。

<canvas>的标准化的努力目前由一个Web浏览器厂商的非正式协会在进行推进，目前<canvas>已经成为 HTML 5 草案中一个正式的标签。

## 2.5 WebSocket

在浏览器中通过HTTP仅能实现单向通信。轮询可以在一定程度上模拟双向通信,但效率比较低，并且需要服务器提供较好的支持才行；Flash中的socket和xmlsocket可以实现真正意义上的双向通信,通过 Flex Ajax Bridge。如果WebSocket在浏览器中得到实现,将会替代以上两项技术，从而得到广泛的使用。面对这种状况，HTML5定义了WebSocket协议，能更好的节省服务器资源和带宽并且达到真正意义上的实时通讯。

WebSocket是HTML5标准中一种新的协议。它是实现了浏览器与服务器全双工通信(Full-Duplex)的接口。现在很多网站为了实现实时通讯(real-time)，所用的技术都是轮询(polling)。轮询是指在特定的的时间间隔(time interval)（如每1秒），由浏览器向服务器发出HTTP请求，然后由服务器返回最新的数据给客服端的浏览器的一种技术解决方案。这种传统的HTTP请求模式带来了明显的缺点，那就是浏览器需要不断的向服务器发出请求(request)，然而HTTP request 的header是非常长的，但里面包含的数据可能只是一个很小的值，这样就会占用很多不必要的带宽。

通过WebSocket API，浏览器和服务器只需要做一个握手的动作，然后，浏览器和服务器之间就形成了一条快速通道，两者之间就直接可以数据实时互相传送。

# 第 3 章 技术原理

## 3.1 Canvas 标签的使用

### 3.1.1 Canvas 标签概述

Canvas API（画布）用于在网页实时生成图像，并且可以操作图像内容，JavaScript能够调用它来进行绘图，基本上它是一个可以用JavaScript操作的位图（bitmap），它像所有的dom对象一样它有自己本身的属性、方法和事件，其中就有实现绘图方法的接口。

使用前，首先需要新建一个canvas网页元素。

```
<canvas id="myCanvas" width="400" height="200">
    您的浏览器不支持canvas!
</canvas>
```

上面代码中，如果浏览器不支持这个API，则就会显示canvas标签中间的文字：“您的浏览器不支持Canvas！”。

每个canvas元素都有一个对应的上下文对象，称作context对象，Canvas API定义在这个context对象载体之上，所以在使用Canvas的方法之前需要获取这个对象，方法是使用getContext方法。

```
var canvas = document.getElementById('myCanvas');
if (canvas.getContext) {
    var ctx = canvas.getContext('2d');
}
```

上面代码中，getContext方法指定参数2d，表示该canvas对象用于生成2D图案（即平面图案）。如果参数是3d，就表示用于生成3D图像（即立体图案），这部分实际上单独叫做WebGL API（本课题不涉及）。

### 3.1.2 绘图方法

canvas画布提供了一个用来作图的平面空间，该空间的每个点都有自己的坐标，x表示横坐标，y表示竖坐标。原点(0, 0)位于图像左上角，y轴的正向是原点向下，x轴的正向是原点向右。

（1）绘制路径

beginPath方法表示开始绘制路径，moveTo(x, y)方法设置线段的起点，lineTo(x, y)方法设置线段的终点，stroke方法用来给透明的线段着色。

```
ctx.beginPath(); // 开始路径绘制
ctx.moveTo(20, 20); // 设置路径起点
ctx.lineTo(200, 20); // 绘制一条到200, 20的直线
ctx.lineWidth = 1.0; // 设置线宽
ctx.strokeStyle = "#CC0000"; // 设置线的颜色
ctx.stroke(); // 进行线的着色，这时整条线才变得可见
```

moveto和lineto方法可以多次使用。最后，还可以使用closePath方法，自动绘制一条当前点到起点的直线，形成一个封闭图形，省却使用一次lineto方法。

（2）绘制矩形

fillRect(x, y, width, height)的方法被定义用来绘制矩形，它的四个参数分别为矩形左上角顶点的x坐标、矩形左上角定点的y坐标，以及矩形的宽度和高度。fillStyle属性用来设置矩形的填充色。

```
ctx.fillStyle = 'yellow';
ctx.fillRect(50, 50, 200, 100);
```

strokeRect方法与fillRect类似，用来绘制空心矩形。

```
ctx.strokeRect(10,10,200,100);
```

clearRect方法用来清除某个矩形区域的内容。

```
ctx.clearRect(100,50,50,50);
```

（3）绘制文本

fillText(string, x, y) 用来绘制文本，它的三个参数分别为文本内容、起点的x坐标、y坐标。在绘制文本之前，需要使用font设置字体、大小、样式（其写法与CSS的font属性基本一致）。与此类似的还有strokeText方法，但它用来添加空心文字。

```
// 设置字体
ctx.font = "Bold 20px Arial";
// 设置对齐方式
ctx.textAlign = "left";
// 设置填充颜色
ctx.fillStyle = "#008600";
// 设置字体内容，以及在画布上的位置
ctx.fillText("Hello!", 10, 50);
// 绘制空心字
ctx.strokeText("Hello!", 10, 100);
```

fillText方法现阶段是不支持文本断行的，也就是所有绘制出来的文本全部出现在一行内。所以，如果要生成多行文本，就需要调用多次fillText方法以解决不能换行的问题。

（4）绘制圆形和扇形

arc方法用来绘制扇形。

```
ctx.arc(x, y, radius, startAngle, endAngle, anticlockwise);
```

arc绘图方法的x和y参数是圆心坐标，radius是圆半径，startAngle和endAngle则是扇形的起始角度和终止角度（以弧度表示），anticlockwise表示做图时应该逆时针画（true）还是顺时针画（false）。

绘制实心的圆形：

```
ctx.beginPath();
ctx.arc(60, 60, 50, 0, Math.PI*2, true);
ctx.fillStyle = "#000000";
ctx.fill();
```

绘制空心圆形：

```
ctx.beginPath();
ctx.arc(60, 60, 50, 0, Math.PI*2, true);
ctx.lineWidth = 1.0;
ctx.strokeStyle = "#000";
ctx.stroke();
```

（5）设置渐变色

createLinearGradient方法用来设置渐变色。

```
var myGradient = ctx.createLinearGradient(0, 0, 0, 160);
myGradient.addColorStop(0, "#BABABA");
myGradient.addColorStop(1, "#636363");
```

createLinearGradient方法的参数是(x1, y1, x2, y2)，其中x1和y1是起点坐标，x2和y2是终点坐标。通过不同的坐标值，可以生成从上至下、从左到右的渐变等等。

使用方法如下：

```
ctx.fillStyle = myGradient;
ctx.fillRect(10,10,200,100);
```

（6）设置阴影

一系列与阴影相关的方法，可以用来设置阴影。

```
ctx.shadowOffsetX = 10; // 设置水平位移
ctx.shadowOffsetY = 10; // 设置垂直位移
ctx.shadowBlur = 5; // 设置模糊度
ctx.shadowColor = "rgba(0,0,0,0.5)"; // 设置阴影颜色
ctx.fillStyle = "#CC0000";
ctx.fillRect(10,10,200,100);
```

### 3.1.3 图像处理方法

drawImage方法

canvas提供的这个方法允许将图像文件插入画布，实现流程是读取图片后，使用drawImage方法在画布内对图像进行重绘。

```
var img = new Image();
img.src = "image.png";
ctx.drawImage(img, 0, 0); // 设置对应的图像对象，以及它在画布上的位置
```

由于图像的载入需要时间，drawImage方法只能在图像完全载入后才能调用，因此上面的代码需要改写。

```
var image = new Image();
image.onload = function() {
    if (image.width != canvas.width)
        canvas.width = image.width;
    if (image.height != canvas.height)
        canvas.height = image.height;
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    ctx.drawImage(image, 0, 0);
```

```
}
image.src = "image.png";
```

drawImage()方法接受三个参数，第一个参数是图像文件的DOM元素（即img标签），第二个和第三个参数则是表示图像左上角在Canvas元素中的坐标，比如上例中的（0, 0）就表示将图像左上角放置在Canvas元素的左上角。

getImageData方法和putImageData方法

getImageData方法可以用来读取Canvas的内容，返回一个对象，包含了每个像素的信息。

```
var imageData = context.getImageData(0, 0, canvas.width, canvas.height);
```

imageData对象有一个data属性，它的值是一个一维数组。该数组的值，依次是每个像素的红、绿、蓝、alpha通道值，因此该

数组的长度=图像的像素宽度 x 图像的像素高度 x 4，

每个值的范围是0－255。这个数组不仅可读，而且可写，因此通过操作这个数组的值，就可以达到操作图像的目的。修改这个数组以后，使用putImageData方法将数组内容重新绘制在Canvas上。

```
context.putImageData(imageData, 0, 0);
```

toDataURL方法

对图像数据做出修改以后，允许使用toDataURL方法，将Canvas数据重新转化成一般的图像文件形式。

```
function convertCanvasToImage(canvas) {
   var image = new Image();
   image.src = canvas.toDataURL("image/png");
   return image;
}
```

上面的代码将Canvas数据，转化成PNG data URI。

save方法，restore方法

save方法用于保存绘制的上下文环境，restore方法则对应用于恢复到上一次保存的上下文环境。

```
ctx.save();
ctx.shadowOffsetX = 10;
ctx.shadowOffsetY = 10;
ctx.shadowBlur = 5;
ctx.shadowColor = "rgba(0,0,0,0.5)";
ctx.fillStyle = "#CC0000";
ctx.fillRect(10,10,150,100);
ctx.restore();
ctx.fillStyle = "#000000";
ctx.fillRect(180,10,150,100);
```

上面代码先用save方法，保存了当前设置，然后绘制了一个有阴影的矩形。接着，使

用restore方法，恢复了保存前的设置，绘制了一个没有阴影的矩形。

### 3.1.4 像素处理

通过getImageData方法和putImageData方法，可以处理每个像素，进而操作图像内容。

假定filter是一个处理像素的函数，那么整个对Canvas的处理流程，可以用下面的代码表示。

```
if (canvas.width > 0 && canvas.height > 0) {
    var imageData = context.getImageData(0, 0, canvas.width, canvas.height);
    filter(imageData);
    context.putImageData(imageData, 0, 0);
}
```

以下研究几种常见的处理方法。

（1）灰度效果

灰度图（grayscale）就是取红、绿、蓝三个像素值的算术平均值，实际上将图像转化了黑白形式。假定d[i]是像素数组中一个象素的红色值，则d[i+1]为绿色值，d[i+2]为蓝色值，d[i+3]就是alpha通道值。转成灰度的算法是将红、绿、蓝三个值相加后除以3，然后再将结果写回像素数组。

```
grayscale = function (pixels) {
    var d = pixels.data;
    for (var i = 0; i < d.length; i += 4) {
       var r = d[i];
       var g = d[i + 1];
       var b = d[i + 2];
       d[i] = d[i + 1] = d[i + 2] = (r+g+b)/3;
    }
    return pixels;
};
```

（2）复古效果

复古效果（sepia）则是将红、绿、蓝三种像素，分别取这三个值的某种加权平均值，从而使得图像有一种古旧的效果。

```
sepia = function (pixels) {
    var d = pixels.data;
    for (var i = 0; i < d.length; i += 4) {
       var r = d[i];
       var g = d[i + 1];
       var b = d[i + 2];
       d[i]     = (r * 0.393)+(g * 0.769)+(b * 0.189); // red
       d[i + 1] = (r * 0.349)+(g * 0.686)+(b * 0.168); // green
       d[i + 2] = (r * 0.272)+(g * 0.534)+(b * 0.131); // blue
    }
    return pixels;
```

```
};
```

（3）红色蒙版效果

红色蒙版指的是，让图像呈现一种偏红的效果。算法是将红色通道设为红、绿、蓝三个值的平均值，而将绿色通道和蓝色通道都设为0。

```
red = function (pixels) {
    var d = pixels.data;
    for (var i = 0; i < d.length; i += 4) {
      var r = d[i];
      var g = d[i + 1];
      var b = d[i + 2];
      d[i] = (r+g+b)/3;              // 红色通道取平均值
      d[i + 1] = d[i + 2] = 0; // 绿色通道和蓝色通道都设为0
    }
    return pixels;
};
```

（4）亮度效果

亮度效果（brightness）是指让图像变得更亮或更暗。算法将红色、绿色和蓝色三个通道，同时加上一个正值或负值即可。

```
brightness = function (pixels, delta) {
    var d = pixels.data;
    for (var i = 0; i < d.length; i += 4) {
          d[i] += delta;          // red
          d[i + 1] += delta; // green
          d[i + 2] += delta; // blue
    }
    return pixels;
};
```

（5）反转效果

反转效果（invert）是指图片呈现一种色彩颠倒的效果。算法为红、绿、蓝通道都取各自的相反值（255-原值）。

```
invert = function (pixels) {
    var d = pixels.data;
    for (var i = 0; i < d.length; i += 4) {
        d[i] = 255 - d[i];
        d[i+1] = 255 - d[i + 1];
        d[i+2] = 255 - d[i + 2];
    }
    return pixels;
};
```

## 3.2 WebSocket 解读

### 3.2.1 何为 WebSocket

WebSocket是HTML5标准提出的一个协议规范。WebSocket约定了类似这么一个通信的规范，通过一个握手的机制，使得客户端（通常是浏览器）和服务器（web server）之间能建立一个类似TCP的连接，从而方便C/S之间的通信。

在WebSocket出现之前，web交互一般是基于HTTP协议的短轮询或者长轮询。

短轮询的过程大概有下面几个步骤：

（1）建立TCP连接；

（2）浏览器发出HTTP请求；

（3）web server应答；

（4）断开TCP连接；

优点是简洁明了，缺点也很明显，因为HTTP协议的头比较长，所以每一次的交互中，建立和断开TCP连接带来了比较大的开销，同时也带来了大量的带宽浪费。

通过设置HTTP头中的keep－alive域可以实现HTTP长连接，避免了建立和断开连接的开销，但是HTTP协议头的问题仍然无法解决。

在WebSocket出现之前，一般通过两种方式来实现Web实时应用：轮询机制和流技术，其中轮询有不同的轮询。

轮询：这是最早的一种实现实时 Web 应用的方案。客户端以一定的时间间隔向服务端发出请求，以频繁请求的方式来保持客户端和服务器端的同步。这种同步方案的缺点是，当客户端以固定频率向服务 器发起请求的时候，服务器端的数据可能并没有更新，这样会带来很多无谓的网络传输，所以这是一种非常低效的实时方案。

长轮询：是对定时轮询的改进和提高，目地是为了降低无效的网络传输。当服务器端没有数据更新的时候，连接会保持一段时间周期直到数据或状态改变或者 时间过期，通过这种机制来减少无效的客户端和服务器间的交互。当然，如果服务端的数据变更非常频繁的话，这种机制和定时轮询比较起来没有本质上的性能的提高。

流：常就是在客户端的页面使用一个隐藏的窗口向服务端发出一个长连接的请求。服务器端接到这个请求后作出回应并不断更新连接状态以保证客户端和服务 器端的连接不过期。通过这种机制可以将服务器端的信息源源不断地推向客户端。这种机制在用户体验上有一点问题，需要针对不同的浏览器设计不同的方案来改进用户体验，同时这种机制在并发比较大的情况下，对服务器端的资源是一个极大的考验。

上述三类方式其实并不是真正的实时技术，只是使用了一种技巧来模拟实时，也叫伪实时。在每次客户端和服务器端交互的时候都是一次 HTTP 的请求和应答的过程，而每一次的 HTTP 请求和应答都带有完整的 HTTP 头信息，这就增加了每次传输的数据量。但这些方式最痛苦的是开发人员，因为不论客户端还是服务器端的实现都很复杂，为了模拟比较真实的实时效果，开发人员 往往需要构造两个HTTP连接来模拟客户端和服务器之间的双向通讯，一个连接用来处理客户端到服务器端的数据传输，一个连接用来处理服务器端到客户端的数 据传输，这不可避免地增加了编程实现的复杂度，也增加了服务器端

的负载，制约了应用系统的扩展性。

基于上述弊端，实现Web实时应用的技术出现了，WebSocket通过浏览器提供的API真正实现了具备像C/S架构下的桌面系统的实时通讯能力。其原理是使用JavaScript调用浏览器的API发出一个WebSocket请求至服务器，经过一次握手，和服务器建立了TCP通讯，因为它本质上是一个TCP连接，所以数据传输的稳定性强和数据传输量比较小。

### 3.2.2 WebSocket 握手协议

WebSocket是一种全新的协议，不属于HTTP无状态协议，协议名为"ws"，这也就代表着一个WebSocket连接地址的写法通常是这样的：ws://**。

WebSocket协议本质上是一个基于TCP的协议。建立连接需要握手，客户端（浏览器）首先向服务器（web server）发起一条特殊的HTTP请求，web server解析后生成应答到浏览器，这样子一个WebSocket连接就建立了，直到某一方关闭连接。

由于WebSocket是草案这个原因，前后便出现了多个版本的握手协议，我在这里分情况说明一下：

（1） 基于Flash的握手协议

使用场景大多是IE的版本，因为IE的多数版本不都不支持WebSocket协议，以及FF、CHROME等浏览器的较低版本，都还没有原生的支持WebSocket。因此这里，server唯一要做的就是准备一个WebSocket-Location域给相应的客户端，没有加密，可靠性很差。

客户端请求：

GET /ls HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
Host: www.myqianlan.com
Origin: HTTP://www.myqianlan.com

服务器返回：

HTTP/1.1 101 Web Socket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: HTTP://www.myqianlan.com
WebSocket-Location: ws://www.myqianlan.com/ls

（2）基于md5加密方式的握手协议

客户端请求：

GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
Sec-WebSocket-Key2: **
Upgrade: WebSocket
Sec-WebSocket-Key1: **
Origin: HTTP://www.myqianlan.com
[8-byte security key]

服务端返回：

HTTP/1.1 101 WebSocket Protocol Handshake

Upgrade: WebSocket

Connection: Upgrade

WebSocket-Origin: HTTP://www.qianlan.com

WebSocket-Location: ws://example.com/demo

[16-byte hash response]

其中 Sec-WebSocket-Key1，Sec-WebSocket-Key2 和 [8-byte security key] 这几个头信息是web server用来生成应答信息的来源，依据draft-hixie-theWebSocketprotocol-76 草案的定义。

web server使用下面的的算法来发出正确的应答信息：

1）首先是逐个字符读取 Sec-WebSocket-Key1 头信息中的所有值，然后将数值型字符连接到一起放到一个临时字符串里，并且在这里同时统计所有空格的数量；

2）将在第（1）步里生成的数字字符串转换成一个整型数字，然后除以第（1）步里统计出来的空格数量，并且将得到的浮点数转换成整数型；

3）将第（2）步里生成的整型值转换为符合网络传输的网络字节数组；

4）对 Sec-WebSocket-Key2 头信息同样进行第（1）到第（3）步的操作，得到另外一个网络字节数组；

5）将 [8-byte security key] 和在第（3）、（4）步里生成的网络字节数组合并成一个16字节的数组；

6）对第（5）步生成的字节数组使用MD5算法生成一个哈希值，这个哈希值就作为安全密钥返回给客户端，以表明服务器端获取了客户端的请求，同意创建WebSocket连接

（3） 基于sha加密方式的握手协议

这是目前见的最多的一种方式。

客户端请求：

GET /ls HTTP/1.1

Upgrade: WebSocket

Connection: Upgrade

Host: www.myqianlan.com

Sec-WebSocket-Origin: HTTP://www.myqianlan.com

Sec-WebSocket-Key: 2SCVXUeP9cTjV+0mWB8J6A==

Sec-WebSocket-Version: 13

服务器返回：

HTTP/1.1 101 Switching Protocols

Upgrade: WebSocket

Connection: Upgrade

Sec-WebSocket-Accept: mLDKNeBNWz6T9SxU+o0Fy/HgeSw=

这个的原理，就是把客户端上报的key拼上一段GUID

“258EAFA5-E914-47DA-95CA-C5AB0DC85B11″，然后拿这个字符串做SHA-1 hash计算，

再然后把得到的结果通过base64方式进行加密，最后返回给相应的客户端。

### 3.2.3 WebSocket 数据帧

WebScoket协议中，数据以帧序列的形式传输。

（1）客户端向服务器传输的数据帧必须进行掩码处理：服务器若接收到未经过掩码处理的数据帧，则必须主动关闭连接。

（2）服务器向客户端传输的数据帧一定不能进行掩码处理。客户端若接收到经过掩码处理的数据帧，则必须主动关闭连接。

针对上述两种情况，发现错误的一方可向对方发送close帧（状态码是1002，表示协议错误），从而关闭这个WebSocket连接。

FIN：1位

表示这是消息的最后一帧（也就是结束帧），一个消息是由一个或多个数据帧构成。若消息由一帧构成，这种特殊情况起始帧就是结束帧。

RSV1，RSV2，RSV3：各1位

如果未定义扩展，各位是0；如果定义了扩展，即为非0值。如果接收的帧此处非0，扩展中却没有该值的定义，那么就关闭这个连接。

OPCODE：4位

这个是定义用来解释PayloadData，如果接收到未知的opcode，接收端就必须关闭连接。

0×0表示附加数据帧

0×1表示文本数据帧

0×2表示二进制数据帧

0×3-7暂时无定义，为以后的非控制帧保留

0×8表示连接关闭

0×9表示ping

0xA表示pong

0xB-F暂时无定义，为以后的控制帧保留

MASK：1位

用于标识PayloadData是否经过掩码处理。如果是1，Masking-key域的数据即是掩码密钥，用于解码PayloadData。客户端发出的数据帧需要进行掩码处理，所以此位是1。

Payload length：7位，7+16位，7+64位, PayloadData的长度（以字节为单位）。

(1) 如果其值在0-125，则是payload的真实长度。

(2) 如果值是126，则后面2个字节形成的16位无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。

(3) 如果值是127，则后面8个字节形成的64位无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。

长度表示遵循一个原则，用最少的字节表示长度。比如payload真实长度是124，在0-125之间，必须用前7位表示；不允许长度1是126或127，然后长度2是124，这样违反原则。

### 3.2.4 客户端 API

客户端本身是由浏览器提供了API，所以我门使用JavaScript来进行调用便可以实现了。

WebSocket JavaScript 接口定义:

```
[Constructor(in DOMString url, optional in DOMString protocol)]
interface WebSocket {
    readonly attribute DOMString URL;
    // ready state
    const unsigned short CONNECTING = 0;
    const unsigned short OPEN = 1;
    const unsigned short CLOSED = 2;
    readonly attribute unsigned short readyState;
    readonly attribute unsigned long bufferedAmount;
    // networking
    attribute Function onopen;
    attribute Function onmessage;
    attribute Function onclose;
    boolean send(in DOMString data);
    void close();
};
WebSocket implements EventTarget;
```

简单了解下接口方法和属性：

readyState表示连接有四种状态：

CONNECTING (0)：表示还没建立连接；

OPEN (1)： 已经建立连接，可以进行通讯；

CLOSING (2)：通过关闭握手，正在关闭连接；

CLOSED (3)：连接已经关闭或无法打开；

url是代表了WebSocket服务器的网络地址，协议一般情况下是”ws”或“wss(加密通信)”，send 方法就是发送数据到服务器端；

close 方法就是关闭连接；

onopen连接建立，即握手成功触发的事件；

onmessage收到服务器消息时触发的事件；

onerror异常触发的事件；

onclose关闭连接触发的事件；

JavaScript调用浏览器接口实例如下：

```
var wsServer = 'ws://localhost:8888/Demo'; //服务器地址
var WebSocket = new WebSocket(wsServer); //创建WebSocket对象
WebSocket.send("hello");//向服务器发送消息
alert(WebSocket.readyState);//查看WebSocket当前状态
WebSocket.onopen = function (evt) {
```

```
    //已经建立连接
};
WebSocket.onclose = function (evt) {
    //已经关闭连接
};
WebSocket.onmessage = function (evt) {
    //收到服务器消息，使用evt.data提取
};
WebSocket.onerror = function (evt) {
    //产生异常
};
```

### 3.2.5 服务器处理

握手协议的客户端数据已经由浏览器自带了，服务器端需要由我们自己来实现。

服务器端需要根据协议来握手、接收和发送。

握手

握手协议：

客户端发到服务器的内容:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: HTTP://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

从服务器到客户端的内容:

```
HTTP/1.1 101 Switching Protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

关键是服务器端Sec-WebSocket-Accept，它是根据Sec-WebSocket-Key计算出来的：

取出Sec-WebSocket-Key，与一个magic string “258EAFA5-E914-47DA-95CA-C5AB0DC85B11” 连接成一个新的key串；将新的key串SHA1编码，生成一个由多组两位16进制数构成的加密串；然后把加密串进行base64编码生成最终的key，这个key就是我们需要的Sec-WebSocket-Key；

代码如下：

```
/// <summary>
///  生成Sec-WebSocket-Accept
```

```
/// </summary>
/// <param name="handShakeText">客户端握手信息</param>
/// <returns>Sec-WebSocket-Accept</returns>
private static string GetSecKeyAccetp(byte[] handShakeBytes,int bytesLength)
{
    string handShakeText = Encoding.UTF8.GetString(handShakeBytes, 0, bytesLength);
    string key = string.Empty;
    Regex r = new Regex(@"Sec\-WebSocket\-Key:(.*?)\r\n");
    Match m = r.Match(handShakeText);
    if (m.Groups.Count != 0)
    {
   key = Regex.Replace(m.Value, @"Sec\-WebSocket\-Key:(.*?)\r\n", "$1").Trim();
    }
    byte[] encryptionString = SHA1.Create().ComputeHash(Encoding.ASCII.GetBytes(key + "258EAFA5-E914-47DA-95CA-C5AB0DC85B11"));
    return Convert.ToBase64String(encryptionString);
}
```

如果握手成功，将会触发客户端的onopen事件。

解析接收的客户端信息

接收到客户端数据解析规则如下：

1byte

1bit: frame-fin，x0表示该message后续还有frame；x1表示是message的最后一个frame

3bit: 分别是frame-rsv1、frame-rsv2和frame-rsv3，通常都是x0

4bit: frame-opcode，x0表示是延续frame；x1表示文本frame；x2表示二进制frame；x3-7保留给非控制frame；x8表示关 闭连接；x9表示ping；xA表示pong；xB-F保留给控制frame

2byte

1bit: Mask，1表示该frame包含掩码；0，表示无掩码

7bit、7bit+2byte、7bit+8byte: 7bit取整数值，若在0-125之间，则是负载数据长度；若是126表示，后两个byte取无符号16位整数值，是负载长度；127表示后8个 byte，取64位无符号整数值，是负载长度

3-6byte: 这里假定负载长度在0-125之间，并且Mask为1，则这4个byte是掩码

7-end byte: 长度是上面取出的负载长度，包括扩展数据和应用数据两部分，通常没有扩展数据；若Mask为1，则此数据需要解码，解码规则为1-4byte掩码循环和数据byte做异或操作。

解析代码如下，但没有处理多帧和不包含掩码的包：

```
/// <summary>
///  解析客户端数据包
/// </summary>
/// <param name="recBytes">服务器接收的数据包</param>
```

```
/// <param name="recByteLength">有效数据长度</param>
/// <returns></returns>
private static string AnalyticData(byte[] recBytes, int recByteLength)
{
    if (recByteLength < 2) { return string.Empty; }
    bool fin = (recBytes[0] & 0x80) == 0x80; // 1bit，1表示最后一帧
    if (!fin){
  return string.Empty;// 超过一帧暂不处理
    }
    bool mask_flag = (recBytes[1] & 0x80) == 0x80; // 是否包含掩码
    if (!mask_flag){
  return string.Empty;// 不包含掩码的暂不处理
    }
    int payload_len = recBytes[1] & 0x7F; // 数据长度
    byte[] masks = new byte[4];
    byte[] payload_data;
    if (payload_len == 126){
  Array.Copy(recBytes, 4, masks, 0, 4);
  payload_len = (UInt16)(recBytes[2] << 8 | recBytes[3]);
  payload_data = new byte[payload_len];
  Array.Copy(recBytes, 8, payload_data, 0, payload_len);
    }else if (payload_len == 127){
  Array.Copy(recBytes, 10, masks, 0, 4);
  byte[] uInt64Bytes = new byte[8];
  for (int i = 0; i < 8; i++){
      uInt64Bytes[i] = recBytes[9 - i];
  }
  UInt64 len = BitConverter.ToUInt64(uInt64Bytes, 0);
  payload_data = new byte[len];
  for (UInt64 i = 0; i < len; i++){
      payload_data[i] = recBytes[i + 14];
  }
    }else{
  Array.Copy(recBytes, 2, masks, 0, 4);
  payload_data = new byte[payload_len];
  Array.Copy(recBytes, 6, payload_data, 0, payload_len);
    }
    for (var i = 0; i < payload_len; i++){
  payload_data[i] = (byte)(payload_data[i] ^ masks[i % 4]);
    }
    return Encoding.UTF8.GetString(payload_data);
```

```
}
```

发送数据至客户端

服务器发送的数据以0x81开头，紧接着的是需要发送内容的长度（若长度在0-125，则1个byte表示长度；若长度不超过0xFFFF，则后2个byte 作为无符号16位整数表示长度；若超过0xFFFF，则后8个byte作为无符号64位整数表示长度），最后是内容的byte数组。

代码如下：

```
/// <summary>
/// 打包服务器数据
/// </summary>
/// <param name="message">数据</param>
/// <returns>数据包</returns>
private static byte[] PackData(string message)
{
    byte[] contentBytes = null;
    byte[] temp = Encoding.UTF8.GetBytes(message);
    if (temp.Length < 126){
    contentBytes = new byte[temp.Length + 2];
    contentBytes[0] = 0x81;
    contentBytes[1] = (byte)temp.Length;
    Array.Copy(temp, 0, contentBytes, 2, temp.Length);
    }else if (temp.Length < 0xFFFF){
    contentBytes = new byte[temp.Length + 4];
    contentBytes[0] = 0x81;
    contentBytes[1] = 126;
    contentBytes[2] = (byte)(temp.Length & 0xFF);
    contentBytes[3] = (byte)(temp.Length >> 8 & 0xFF);
    Array.Copy(temp, 0, contentBytes, 4, temp.Length);
    }else{
    // 暂不处理超长内容
    }
    return contentBytes;
}
```

### 3.2.6 目前主流的浏览器支持情况

目前主流浏览器对WebSocket的支持情况如表3-1所示。

表 3-1 主流浏览器对 WebSocket 的支持情况

| IE | Firefox | Chrome | Safari | Opera |
|---|---|---|---|---|
| 10+ | 6+ | 14+ | 5+ | 10+ |

# 第 4 章 实际应用开发

## 4.1 应用设计

### 4.1.1 应用简介

利用HTML5的canvas和WebSocket等，以及Linux与nodejs作为服务器支撑，构建的一个Web实时交互应用，该应用跨多终端多平台多浏览器。应用为用户打开一个浏览器页面，即可运用相关操作（鼠标点击或者触摸），在页面创造多姿多彩的烟花，并且可以开启增强模式，这样就可以将多个烟花以图案的形式绘制出来。这些操作的多终端实时共享的，每个终端的页面均会显示当前和你一起正在创造烟花的人数与服务器记录的所有人累计一共创造的烟花总数。

### 4.1.2 应用原型界面

应用界面如图4-1至4-9所示。


图 4-1 开始界面


4-2 应用主界面

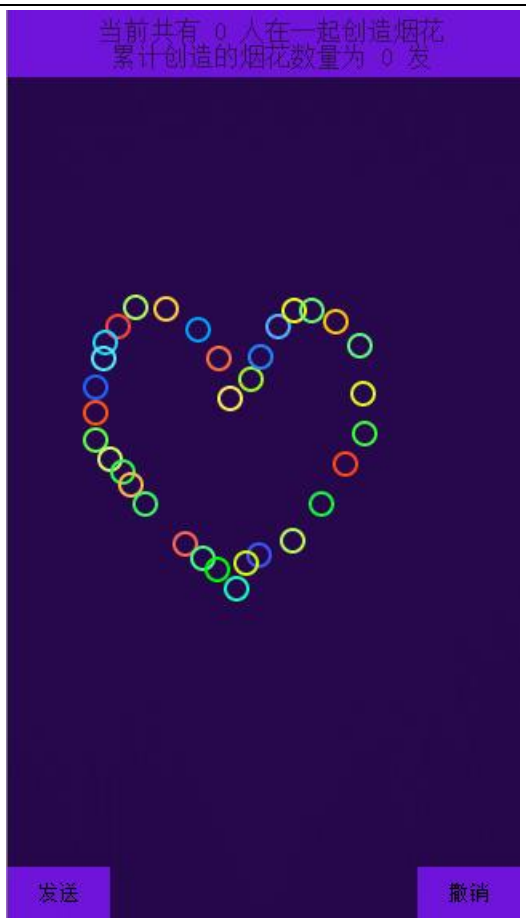
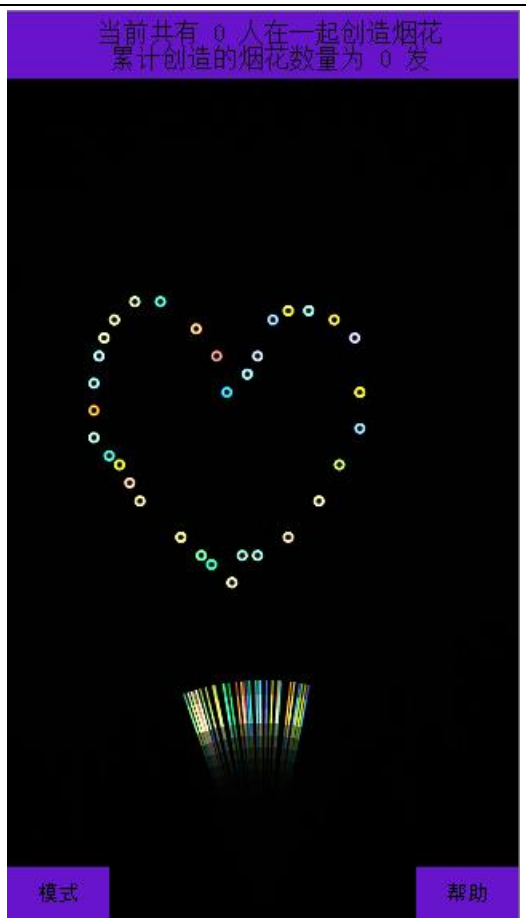



图 4-3 增强模式界面




4-4 增强模式效果界面


图 4-5 发送绘图确认框


4-6 发送绘图失败提示框


图 4-7 开始界面图


4-8 应用主界面


图 4-9 帮助界面

## 4.2 应用系统逻辑

服务器与客户端初次握手时的流程如图4-10所示。

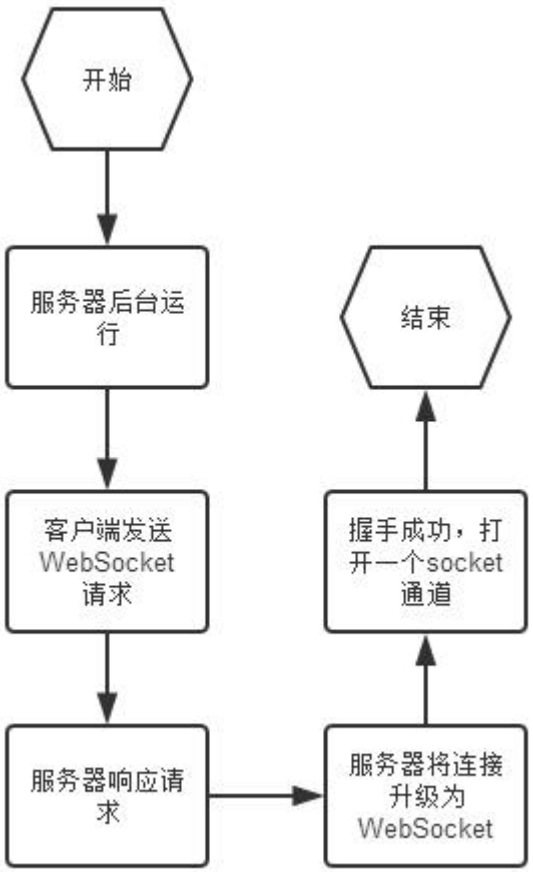



图 4-10 服务器与客户端初次握手

服务器与任何一台客户端的通信都是基于WebSocket，都是实时的，都是并发过程。并且，所有的事件处理都是异步执行，互不干扰。这里，选取其中一个实时通信的事件，其流程图如图4-11所示。

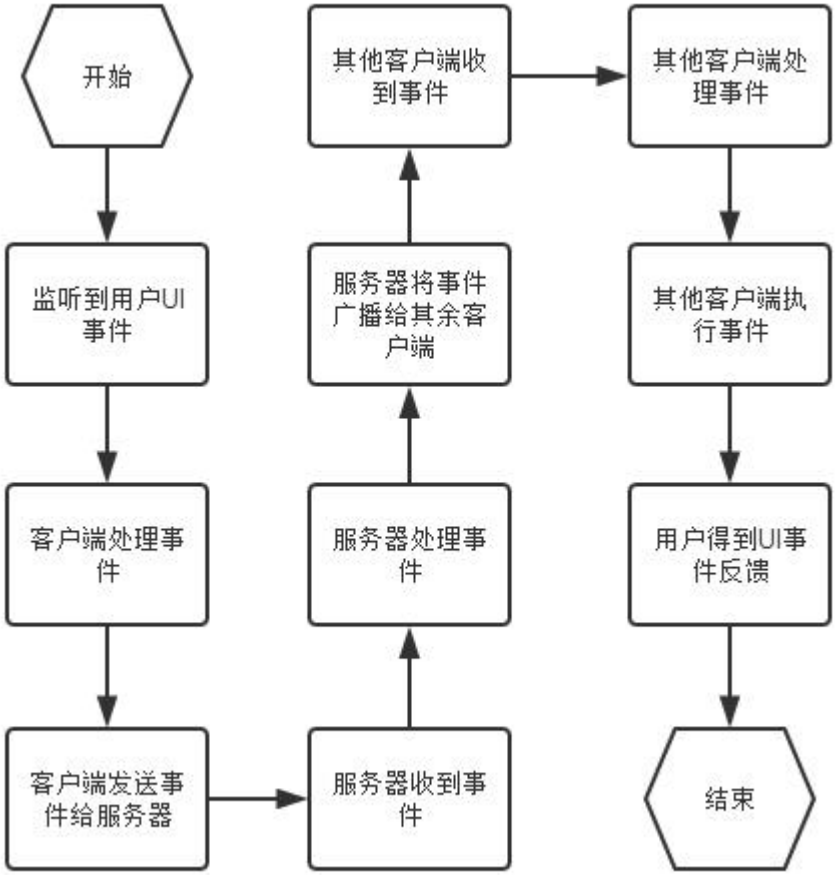



图 4-11 服务器与客户端的单次实时通信

## 4.3 应用实现

### 4.3.1 客户端绘制烟花

烟花效果实际上是一个朝一个点位移动画加上爆炸例子效果。本应用使用请求重绘帧（requestAnimationFrame）来实现烟花的动画效果。

绘制关键代码如下：

```
// draw firework
Firework.prototype.draw = function() {
    hue = random(0, 255);
    mainCanvasCtx.beginPath();
    // move to the last tracked coordinate in the set, then draw a line to the current x and y
    mainCanvasCtx.moveTo(this.coordinates[this.coordinates.length - 1][0], this.coordinates[this.coordinates.length - 1][1]);
    mainCanvasCtx.lineTo(this.x, this.y);
    mainCanvasCtx.strokeStyle = "hsl(" + hue + ", 100%, " + this.brightness + "%)";
    mainCanvasCtx.stroke();

    mainCanvasCtx.beginPath();
    // draw the target for this firework with a pulsing circle
```

```
            mainCanvasCtx.arc(this.tx, this.ty, this.targetRadius, 0, Math.PI * 2);
            mainCanvasCtx.stroke();
        }


        // draw particle
        Particle.prototype.draw = function() {
            mainCanvasCtx.beginPath();
            // move to the last tracked coordinates in the set, then draw a line to the current x and y
            mainCanvasCtx.moveTo(this.coordinates[this.coordinates.length - 1][0], this.coordinates[this.coordinates.length - 1][1]);
            mainCanvasCtx.lineTo(this.x, this.y);
            mainCanvasCtx.strokeStyle = "hsla(" + this.hue + ", 100%, " + this.brightness + "%, " + this.alpha + ")";
            mainCanvasCtx.stroke();
        }
```

### 4.3.2 客户端增强模式实现

客户端的增强模式，我们需要设置一个开关，表明我们开启了增强模式。

Var modeToggle = false;默认为fasle，表示关闭增强模式。

开启增强模式过后，便是用户自定义绘制内容，并且把自定义内容的所有坐标存到一个容器里面，等到用户发送的时候再传递给服务器。

代码如下

```
$drawMode.on("click", function(event) {
        event.preventDefault();
        var x = event.pageX - $drawMode[0].offsetLeft;
        var y = event.pageY - $drawMode[0].offsetTop;
        posList.push({
            mousex: Math.floor(100 * x / cw) / 100,
            mousey: Math.floor(100 * y / ch) / 100
        });
        drawCircle(x, y, drawModeCtx);
    });

    $drawMode.on("touchstart", function(event) {
        event.preventDefault();
        var touches = event.originalEvent.targetTouches;
        var x = touches[0].pageX - $drawMode[0].offsetLeft;
        var y = touches[0].pageY - $drawMode[0].offsetTop;
        posList.push({
            mousex: Math.floor(100 * x / cw) / 100,
```

```
            mousey: Math.floor(100 * y / ch) / 100
        });
        drawCircle(x, y, drawModeCtx);
    });
    //when draw mode is on ,draw a circle
    function drawCircle(x, y, ctx) {
        var brightness = random(50, 70);
        hue = random(0, 255);
        ctx.strokeStyle = "hsl(" + hue + ", 100%, " + brightness + "%)";
        ctx.lineWidth = 2;
        ctx.beginPath();
        ctx.arc(x, y, 7, 0, Math.PI * 2);
        ctx.stroke();
    }
```

### 4.3.3 客户端与服务器端事件

每个客户端都会发送自己普通绘制事件，增强模式绘制事件，以及连接服务器成功的事件。这些实践在服务器都会有相应的监听器。同样，服务器端也会根据相应的情况监听或发送事件给连接着的客户端。

客户端：

```
socket.on("connect", function() {
        /* action when conneted */
        modeConnect = true;
    });

    socket.on("disconnect", function() {
        /* action when conneted */
        modeConnect = false;
        easyDialog.open({
            container: "Error"
        });
    });

    //client change eventlistener
    socket.on("clientChange", function(data) {
        $clientInfo.html(data["clientNum"]);
        $fireworkInfo.html(data["fireworkNum"]);
    });

    //other click eventlistener
    socket.on("otherClick", function(data) {
        mx = cw * data["mousex"];
```

```
            my = ch * data["mousey"];
            $fireworkInfo.html(data["fireworkNum"]);
            createFirework(mx, my);
        });

        //other draw event listener
        socket.on("otherDrawClick", function(data) {
            $fireworkInfo.html(data["fireworkNum"]);
            createDrawFirework(data["posArr"])
    });
    socket.emit("myClick", {
                        mousex: Math.floor(100 * mx / cw) / 100,
                        mousey: Math.floor(100 * my / ch) / 100
                    });
    socket.emit("myDrawClick", posList);
```

服务器端：

```
    var clientNum = 0;
    var fireworkNum = 0;
    io.sockets.on("connection", function(socket) {
        clientNum++;
        console.log("curent cient is : " + clientNum);
        io.sockets.emit("clientChange", {
            clientNum: clientNum,
            fireworkNum: fireworkNum
        });
        //user leave
        socket.on("disconnect", function() {
            clientNum--;
            console.log("one leave");
            socket.broadcast.emit("clientChange", {
                clientNum: clientNum,
                fireworkNum: fireworkNum
            });
        });
        socket.on("myClick", function(data) {
            fireworkNum++;
            console.log(data + "fireworkNum" + fireworkNum);
            io.sockets.emit("otherClick", {
                mousex: data["mousex"],
                mousey: data["mousey"],
                fireworkNum: fireworkNum
            });
```

```
        });
        socket.on("myDrawClick", function(data) {
            fireworkNum += data.length;
            console.log(data + "fireworkNum" + fireworkNum);
            io.sockets.emit("otherDrawClick", {
                posArr: data,
                fireworkNum: fireworkNum
            });
        });
    });
```

## 4.4 应用部署

### 4.4.1 本地部署

在没有服务器的情况下，可以在搭建本地服务器进行测试。

本地服务器文件核心代码：

```
//创建一个HTTP服务器
var webSvr = HTTP.createServer(webServicer),
    io = require("socket.io").listen(webSvr);
//指定服务器错误事件响应
webSvr.on("error", function(error) {
    console.log(error); //在控制台中输出错误信息
});
//开始侦听5000端口
webSvr.listen(5000, function() {
    //向控制台输出服务启动的信息
    console.log("WebSur Start running at localhost:3000");
});
```

在Linux环境下面，我们打开终端（快捷键Ctrl+Alt+T），进入到项目目录，输入：

```
node server.js
```

可以看到在控制台输出了本地服务器启动成功的信息。

然后，打开一个浏览器窗口，在地址栏里面键入：localhost:5000，便能够打开这个实时Web应用。

### 4.4.2 部署到网络生产环境

根据服务器环境的不同，部署方法也不一样。我这里使用的是网络免费服务器，所以性能以及响应上会差一些，但也是能够正常运行的。

部署到的服务器是heroku提供的免费服务器。我们需要在项目下面添加两个文件，package.json和Profile。前者用于提供应用信息，应用环境支持信息等，后者则用于启动服务器需要使用的命令。其中，Profile里面的代码为：

```
web : node server.js
```

文件添加完成过后，通过git（版本管理软件）推送到heroku上面就完成了部署。部署

过后，打开提供的网址便可以了。

# 第 5 章 结论与展望

## 5.1 结论

通过本课题对新一代无插件交互式 Web 技术的理论研究与实际应用，对新一代 Web 交互的实现方式，特别是实时 Web 交互，有了更深层次的理解。对 canvas 标签、WebSocket 全双工实时通信以及 nodejs 服务端等技术也有了比较深入的认识。期间不断的学习新的知识，以满足课题研究的需要，这也将是人生一笔巨大的财富。

## 5.2 不足之处及未来展望

无插件 Web 交互目前因为浏览器的原因还不能够大规模广泛的使用。其一是各个浏览器提供的 api 没有一个统一的标准，导致实现的效果经常会出现各种各样的错误。其二是目前阶段浏览器的性能还比较低下，还不足以支撑强大的应用。

但是随着科技的快速发展，互联网的飞速进步，W3C 浏览器全球规范的推进，前面所提到的问题也都将不再是问题。无插件 Web 交互也将会带来新的交互革命，拭目以待。

# 参考文献

[1] Nicholas C.Zakas 著，李松峰，曹力 译．JavaScript 高级程序设计(第三版)[M]．北京：人民邮电出版社，2013．

[2] Douglas Crockford 著，赵泽欣，学 译. JavaScript 语言精粹（修订版）[M]． 北京：电子工业出版社，2013．

[3] 郭家宝．Node.js 开发指南[M]．北京：人民邮电出版社，2012．

[4] David Flanagan 著，淘宝前端团队 译． JavaScript 权威指南（第六版）[M]．北京：机械工业出版社，2012.

[5] David Gourley，Brian Totty，Marjorie Sayer，Sailu Reddy，Anshu Aggarwal 著，陈涓，赵振平 译．HTTP 权威指南[M]． 北京：人民邮电出版社，2012．

[6] Mike Cantelon，TJ Holowaychuk 著，吴海星 译．Node.js 实战[M]．北京：人民邮电出版社, 2014．

[7] Rob Hawkes 著，周广新等 译． HTML5 Canvas 基础教程[M]．北京：人民邮电出版社，2012．

[8] Vanessa Wang．HTML5 WebSocket 权威指南[M]．北京:机械工业出版社，2014．

[9] Liu Qiqiang，Sun Xiangyang．Research of Web Real-time Communication Based on WebSocket[C] ．Proceedings of 2011 World Congress on Engineering and Technology（CET2011）VOL07,ShangHai：[s.n.]，2011．

[10] Kai Shuang，Feng Kai．Research on Server Push Methods in Web Browser based Instant Messaging Applications[J] ．Journal of Software，2013，Vol.8(10)：2644-2651．

[11] Anonymous．Kaazing Announces Kaazing WebSocket Gateway—the Industry’s First Enterprise – Grade Web Infrastructure Software Based on the HTML5 WebSocket Standard[C] ．Technology&Business Journal，2010．

# 致　谢

本论文是在夏鸿斌老师的悉心指导下完成的，从论文选题、技术研究内容与方向、应用的设计与实现、论文写作、初稿修改、直至最后定稿，夏老师都给了我很多的指导和帮助，在此我要向夏老师由衷的说声谢谢！还要感谢在此次设计中所有帮助过我的老师和同学，以及通过互联网给予我帮助的各位技术同仁。