计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(10):280−284 [doi: 10.15888/j.cnki.csa.006000]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Node.js 的移动视频监控系统①

房　峰, 高美凤

(江南大学 物联网工程学院, 无锡 214122)

摘　要: 使用 OpenCV 和 jQuery Mobile 设计了一款基于 Node.js 运行平台的移动视频监控系统. 系统采用 B/S 结构, 在 windows 系统上搭建的 Node.js 服务器用于接收和发送视频, 同样部署在 windows 系统上的 OpenCV 负责图像的处理和编码;用 jQuery Mobile 结合 Hybrid App 开发模式制作的客户端运行在移动终端, 提供监控功能. 测试结果表明, 在 WiFi 环境下, 系统可进行有效的多用户实时监控.

关键词: 视频监控; jQuery Mobile; Hybrid App; OpenCV; Node.js

引用格式: 房峰,高美凤.基于 Node.js 的移动视频监控系统.计算机系统应用,2017,26(10):280–284. http://www.c-s-a.org.cn/1003-3254/6000.html

Mobile Video Monitoring System Based on Node.js

FANG Feng, GAO Mei-Feng

(School of IoT Engineering, Jiangnan University, Wuxi 214122, China)

Abstract: A mobile video monitoring system based on Node.js is designed by using OpenCV coupling with Web front-end technology in this paper. The system adopts B/S structure. Node.js server is built on windows system for acquisition and transmission of video data and OpenCV built on the same system is used for image processing and encoding. The client is made with Web front-end technology and Hybrid App development model running on the mobile terminal, to provide monitoring function. The test results show that under WiFi environment, the system carries out multi-user and real-time monitoring effectively.

Key words: video monitoring; jQuery Mobile; Hybrid App; OpenCV; Node.js

随着摄像头安装数量的日益增多, 以及智慧城市和公共安全需求的日益增长, 传统的人工视频监控方式已经远远不能满足监控需要, 因此, 智能视频监控技术应运而生并迅速成为一个研究热点[1]. 文献[2]中提出了一种图像传输和分发系统的设计方法, 该系统利用 TCP/IP 协议接收智能视频监控系统输出的图像数据, 再利用 websocket 协议把 data URI 格式的图像数据转发给客户端. 本文利用文献[2]中的相关技术, 在 windows 系统上, 基于 Node.js 运行平台, 采用 Hybrid App 移动开发模式, 结合 OpenCV 图像处理库和 Web 前端技术, 开发了一款移动端低成本多用户实时监控系统.

1　系统结构

移动视频监控系统主要由 USB 摄像头、视频图像处理系统、Node.js 服务器和移动客户端组成, 如图 1 所示.

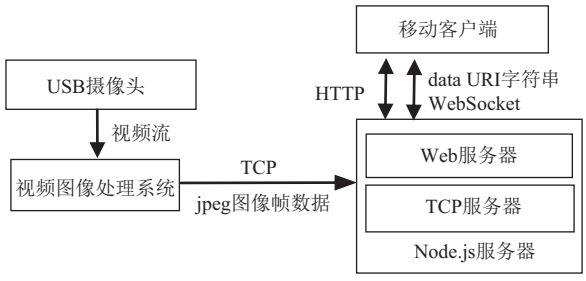

图 1　系统总体结构

① 收稿时间: 2017-01-03; 采用时间: 2017-02-20

其中, Node.js 服务器通过 TCP 协议接收经视频图像处理系统处理过的视频图像数据, 再利用 WebSocket 协议把 jpeg 图像原始二进制数据进行 base64 编码构造的 data URI 字符串转发给移动客户端, 客户端负责接收该字符串并通过 image 元素引用完成视频图像显示功能. 本系统在同一局域网下已经测试成功.

2 视频图像处理系统

OpenCV 提供了非常丰富的帧提取函数和视觉处理算法, 开发者可以在其视频开发项目中直接调用进行算法移植并添加自己的程序, 即可完成复杂庞大的开发任务, 达到事半功倍的效果[3]. 本文视频图像处理系统就是利用 OpenCV 库, 结合多线程方式, 将处理过的视频数据进行 JPEG 压缩, 最终通过 TCP 协议转发给服务器.

2.1 多线程处理视频数据

采用多线程可以提高资源的利用率和程序的响应速度, 使程序设计变得简单. 图像处理部分就是利用基于 windows 系统的 C++多线程编程方法, 完成 3 个基本任务, 即图像获取、图像处理和处理后图像数据的网络发送. 在程序设计上, 使用 3 个线程实现以上 3 个任务, 在线程之间采用互斥锁的机制对共享的关键数据进行保护和实现线程之间的同步, 从而保证系统平稳、流畅运行, 具体的实现机制如图 2 所示. 图像获取线程与图像处理线程之间、图像处理线程和图像发送线程之间共同维护了两个缓冲区 A 和 B, 并且这两个缓冲区分别有两个线程拥有权限对其进行操作, 这样就会产生竞争现象, 破坏数据的完整性, 采用互斥锁机制正好能解决以上问题.

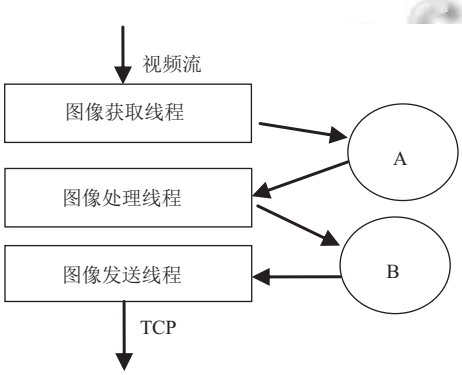

图 2 多线程图像处理机制

2.2 基于 TCP 的视频传输

TCP 是面向连接的通信协议, 提供可靠的数据流服务, 从而确保视频的有效传输. 面向连接的 TCP 协议需要客户端和服务器两个应用. 本文中, 视频图像处理系统作为 TCP Client, 使用 socket 传输经处理后的图像帧数据, 发送给作为 TCP Server 的 Node.js 服务器. 为了保证图像传输的实时性, Node.js 服务器与视频图像处理系统部署在同一台 PC 机上.

2.3 JPEG 图像数据压缩

JPEG 是互联网上使用最广泛的一种图像存储和传送格式[4], 是第一个国际图像压缩标准. JPEG 图像压缩算法能够在提供良好的压缩性能的同时, 具有比较好的重建质量, 被广泛应用于图像、视频处理领域, 使用 JPEG 标准压缩后图像的体积可以减少数十倍. 在图像处理线程中对处理过的图像进行 JPEG 压缩, 将压缩之后的 JPEG 数据再由图像发送线程通过 TCP 协议发送给服务器, 这样就大大减小图像数据体积, 进一步提高数据传输的实时性.

3 Node.js 服务器实现

Node.js 服务器由 TCP 服务器和 Web 服务器组成, 如图 1 所示. 实现的主要功能有 jpeg 图像帧数据的接收, 图像帧数据的编码并构造为 data URI 字符串, 利用 WebSocket 协议进行字符串数据的发送, 处理多用户的 HTTP 请求.

3.1 Node.js 简介

Node.js 是建立在 Chrome V8 之上的运行平台, 它用于构建快速、可扩展的网络应用程序. Node.js 使用一种事件驱动、非阻塞的 I/O 模型, 这也使得跨分布式设备的数据密集型实时应用更加轻量、高效和完美. 并且异步非阻塞的 Node.js 所构建的 Web 应用程序和同步阻塞语言 PHP 相比, 在高并发请求的情况下, Node.js 构建的服务器比 PHP 构建的应用程序响应时间短、吞吐率高[5]. 因此, 本文采用 Node.js 运行平台, 可进一步实现多用户实时监控.

3.2 TCP 服务器

前文所述, 视频数据的传输是基于 TCP 协议的, 视频图像处理系统作为 TCP Client, Node.js 服务器中的 TCP 服务器作为 TCP Server. TCP 服务器主要功能是实现 jpeg 图像帧数据的接收, 因此使用 net 模块提供的异步网络包装器对象来创建 TCP 服务器[6]. 部分代码如下:

```
var net=require("net"); //引用 net 模块
//创建 TCP 服务器
```

```
var socket=net.createServer(function(conn) {
  conn.on('data', function(data){
    //data 事件处理
    });
});
    //监听 4000 端口
socket.listen(4000, "172.18.170.165", function(){
    console.log("正在监听 4000 端口");
});
```

3.3 Web 服务器

HTTP 是 Web 服务器和浏览器使用的通讯协议, Web 服务器用此协议来接收和处理浏览器发出的 HTTP 请求. 使用 Node.js 构建的 Web 服务器除了具有在高并发请求下响应时间短、吞吐率高的性能优势外, 还可以进行路由设置, 从而浏览器可以通过 HTTP 协议很方便地获取服务器上的 html 资源.

Web 服务器主要功能有, JPEG 图像帧数据 base64 编码并构造 data URI 字符串、字符串数据的发送和处理多用户请求.

本系统利用 Node.js 的 Express 框架结合 socket.io 模块搭建 Web 服务器, 部分代码如下:

```
var express=require("express");
var app=express();
var http=require("http").Server(app);
var io=require("socket.io")(http);
...
http.listen(3000, "172.18.170.165");
```

3.3.1 WebSocket

Web 服务器要把图像数据发送给各个客户端, 需要一种实时性较强的全双工通信方式, 而不是单向被动的 HTTP 请求.

WebSocket 是 Html5 的一种新的协议, 它实现了客户端与服务器全双工 socket 通信[7]. 建立 WebSocket 连接后, 双方都可以随时给对方发送数据, 在实时性上比 HTTP 协议更强. 基于这种特点, Web 服务器会将编码构造的字符串数据通过 WebSocket 协议立即转发给与 Web 服务器建立连接的移动客户端, 达到实时转发效果. 在本文实现上, 使用 socket.io 模块实现全双工通信. 部分代码如下:

```
io.on('connection', function(socket){ //监听连接事件
socket.on("start", function(){ //监听客户端事件
...
socket.emit("receive", dataURI);//发送本地事件
...
});});
```

3.3.2 data URI

一般情况下, 在浏览器上通过 image 元素的 src 属性引用图片文件 URL 地址来完成图像的显示. 如果在服务器端把每一帧图像都保存为 jpeg 文件, 再让浏览器通过 URL 引用图像, 这样每一帧图像的引用都要耗费一个 HTTP 请求, 从而造成时间的浪费.

data URI 是由 RFC2397 定义的一种把小文件嵌入文档的方案, 可以有效的减少 HTTP 请求数. 对于图像文件等二进制数据, 可以将文件的二进制数据进行 base64 编码之后再进行嵌入. 针对传输的 JPEG 图像文件, 在 Web 服务器中, 对 JPEG 图像帧数据进行 base64 编码, 并构造成如下字符串:

```
var dataURI=' '+buf.toString
('base64');
```

4 移动客户端实现

客户端软件采用 Hybrid App 移动开发模式和 jQuery Mobile 框架联合编程, 实现了对 data URI 字符串接收、显示、播放和暂停的基本功能. 客户端通过 HTTP 协议接收 Web 服务器返回的 html 文件, 然后通过内嵌在原生应用主窗口中的浏览器内核对该文件进行渲染解析. 其中, data URI 字符串的接收利用 WebSocket 协议实现. 此外, 客户端与服务器之间能够正常使用 WebSocket 协议通信, 除了服务器要引用相应的包之外, 返回给客户端的 html 文件中必须引用以下文件:

```
<script
type="text/javascript"src="/socket.io/socket.io.js">
</script>
```

这样, 才能调用 io 函数, 取得 socket 对象.

4.1 Hybrid App 开发模式和 jQuery Mobile 框架

移动 App 是针对移动设备所开发的应用软件, 智能终端的普及不仅推动了移动互联网的发展, 也带来了移动 App 应用的爆炸式增长. 目前比较流行的有三种移动 App 开发模式: Native App(原生开发模式)、Web App(网页开发模式) 和 Hybrid App(混合开发模式)[8]. 本系统选择混合开发模式, 此模式介于前两者之间, 有效解决了原生应用开发周期长以及网页应用中

用户粘性不够的问题.

jQuery Mobile 是基于 jQuery 和 jQuery UI 的 Web 框架, 专用于移动智能终端平台, 支持多种移动平台. 它具有轻量级的代码, 使用渐进增强方式构建, 具有可伸缩、易更换主题的设计特点[9]. 利用 jQuery Mobile 框架实现前端表现界面不但符合移动用户交互体验, 而且有效解决了移动端兼容问题.

4.2 视频接收与显示

为了实现在客户端上流畅并实时观看经过视频图像处理系统处理后的图像, 采用了在客户端高速而连续引用每一帧图像的 data URI 字符串进行图像显示的方法来模拟视频直播的效果.

移动客户端通过 WebSocket 协议接收经 base64 编码构造的 data URI 字符串, 然后通过 image 元素引用该字符串完成图像显示, 即将接收到的字符串赋值给 image 元素的 src 属性. 此外, 通过一个信号量控制客户端是否接收数据来实现播放与暂停的功能, 部分代码如下:

```
<script type=”text/javascript”>
var socket=io();//调用 io 函数, 获取 socket 对象
var signal; //定义信号量
$(“#start”).on(“tap”, function(){//注册点击事件—开始
    signal=true;
    socket.emit(“start”); //向服务器发送接收信号
});
$(“#stop”).on(“tap”, function(){//注册点击事件—停止
    signal=false;
});
socket.on(“receive”, function(dataURI){//接收服务器端
//传来的 dataURI 字符串
//image 元素引用 dataURI, 实现图像显示
    document.getElementById(“img”).src=dataURI;
    if(signal){//判断信号量决定客户端是否接收数据
        socket.emit(“start”);
    }
});
</script>
```

虽然 jQuery Mobile 是针对移动端的 Web 框架, 解决了控件如按钮对各种移动端屏幕尺寸的适应问题, 但是没有考虑 image 元素的兼容, 导致显示的视频图像大小无法达到自适应屏幕的效果, 要在 head 标签内对 image 元素做如下设置:

```
<head>
...
<style>
#img{width:100%;}
</style>
...
</head>
```

5 系统测试

本系统在 win8 操作系统上进行测试. 视频图像处理系统与 Node.js 服务器部署在同一台 PC 机上, 其中, 视频图像处理应用程序利用 Visual Studio 2013 结合移植入的 OpenCV2 进行编程. Node.js 服务器利用前端开发工具 WebStorm10.0.5 进行开发. 移动客户端运行在 Android 手机上, 利用 ADT 进行开发. 确保在同一可用的 WiFi 环境下后, 先运行 Node.js 服务器, 再运行视频图像处理应用程序, 最后打开手机桌面上的 Monitor 图标如图 3, 按下开始按钮便可实时显示视频图像, 测试结果如图 4 所示.

图 3 Monitor 图标

图 4 系统测试

6 结语

本文介绍了一种基于 Node.js 的移动视频监控系统, 经过测试, 该系统能够实时传输视频图像数据, 让多用户通过移动端进行实时的视频监控, 解决了常规监控系统只显示图像不能对图像进行处理、实时性差和无法多用户同时连接的问题以及传统网页应用中用户粘性不够的问题. 此外, 系统易于搭建, 性能稳定, 成本低廉, 具有广泛的应用价值.

参考文献

1 黄凯奇, 陈晓棠, 康运锋, 等. 智能视频监控技术综述. 计算机学报, 2015, 38(6): 1093–1118. [doi: 10.11897/SP.J.1016.2015.01093]

2 褶润堂. 面向实时视频监控的图像传输及分发系统设计与实现[硕士学位论文]. 北京: 北京邮电大学, 2015.

3 刘瑞祯, 于仕琪. OpenCV 教程—基础篇. 北京: 北京航空航天大学出版社, 2007.

4 冯彦辉, 高洁, 徐畔, 等. 基于 JPEG 图像文件格式的研究. 山西电子技术, 2009, (1): 38–39.

5 王金龙, 宋斌, 丁锐. Node.js: 一种新的 Web 应用构建技术. 现代电子技术, 2015, 38(6): 70–73.

6 陈会安. JavaScript+jQuery Mobile+Node.js 跨平台网页设计. 北京: 机械工业出版社, 2016: 1.

7 肖在昌, 杨文晖, 刘兵. 基于 WebSocket 的实时技术. 电脑与电信, 2012, (12): 40–42. [doi: 10.3969/j.issn.1008-6609.2012.12.034]

8 杨毅. 移动 APP 开发模式探讨. 福建电脑, 2014, 30(6): 86–87.

9 Firtman M. jQuery Mobile: Up and Running. O'Reilly Media Inc., 2012.