基于 Vert.x 的移动平台 Web 实时视频监控①

吕海东

(大连理工大学城市学院 计算机工程分院, 大连 116600)

摘 要: 随着移动互联网的飞速发展, 以及移动手机的普及, 人们对多媒体应用的需求与日俱增. 如何适应未来大量移动客户的高并发连接请求处理以及多媒体应用的视频和音频传输, Vert.x 框架是最佳的解决方案. 基于 Vert.x[1]并结合 HTML5, jQuery 实现了一个在线实时的 Web 多用户视频监控系统, 重点讲述了系统实现技术, 架构设计和编程实现.

关键词: Vert.x; HTML5; 视频监控; 移动应用

Real-Time Web Video Monitoring Based on Vert.x

LV Hai-Dong

(School of Computer Enginerring, City Institute Dalian University of Technology, Dalian 116600, China)

Abstract: With the rapid development of mobile internet and popularity of mobilephone, human increased demand for multimedia applications. Vert.x framework is the best solution to meet the need of high concurrent mobile client connection request processing and multimedia application with large amount of video or audio transmission. The new kind of mobile web video monitoring system was developed with the Vert.x, HTML 5 and jQuery, and the paper mainly descripts the implementation technology, architecture design and system programming.

Key words: Vert.x; HTML5; video moniting; mobile app

随着移动互联网的普及, 以智能手机为监控客户端的移动视频监控系统飞速发展, 必将在各行各业得到广泛应用. 但现有的视频监控系统都是依赖特定手机平台的原生应用, 使用特定平台的编程语言进行开发, 如 Android 平台使用 Java, iPhone 平台使用 Object–C, 无法实现跨所有手机平台运行, 必须为不同的平台开发不同的客户端代码. 另外在视频数据实时传输中依然使用传统的视频流编码技术, 基于视频流点播技术实现监控画面显示, 无法满足大量客户的高并发的场合. 为实现视频监控这种多媒体类型应用, 同时伴随高并发和大数据量传输的需求, 传统的服务器平台技术如 JavaEE, MS.NET, 还是 PHP 都无法满足其高性能和实时性要求. 在此背景下 Vert.x 全新架构的服务器技术应运而生, 其先天性的性能优势, 优异的无阻塞技术, 创新的异步工作模式和事件处理机制, 使之特别适合作为未来高并发连接请求, 大数据传输的移动多媒体应用的运行平台. 同时 HTML5 规范的最终定稿, 预示着 HTML5 将要给移动互联网界带来颠覆性变革, 必将使得 HTML5 的移动 Web 应用全面取代目前流行的本地原生应用. 本文使用 Vert.x 和 HTML5 技术实现了以智能手机和平板为主的视频实时监控系统, 实现实时的多用户群发群收的视频监控, 并工作在 Web 模式下, 实现跨所有移动平台的使用.

1 系统实现技术

1.1 Vert.x 框架

Vert.x 框架是基于 JVM 的轻量级的, 高性能的用于开发现代移动和企业级应用的服务器平台. 其核心特点是多语言, 非阻塞, 单线程, 异步工作模式, 特别适用于开发大量用户连接的高并发应用. Vert.x 与 Node.js[2]采用相同的非阻塞, 异步工作模式, 都是为解决传统的应用服务器, 如 JavaEE, MS.NET, PHP 无法适用当连接超过 10 万的高并发场合. Vert.x 的单线程, 异步, 非阻塞特性特别适合于移动应用, 实时数

① 收稿时间:2015-01-29;收到修改稿时间:2015-03-12

据传输等领域[3].

Vert.x 的最核心组件是分布式事件总线(Distribute Event Bus) [4],基于 Vert.x 的应用系统功能都由组件 Verticle 实现, 但与传统的组件直接调用不同, Verticle 之间不能直接调用, 只能通过事件总线发送命令和数据完成间接的调用, 并且 Verticle 之间的调用是异步的. Verticle 采用单线程模式, 从而可以实现高并发性. 事件总线不但可以在服务器上作为 Verticle 之间的传输通道, 甚至可以延伸到 Web 客户端, 从而实现客户端与服务器的通讯, 甚至实现客户端之间的直接数据传输, 并且实现机制非常简单, 编程容易, 在实现视频传输方面要比流行的 WebRTC 要简单得多. 而 WebRTC 只能实现二个客户端之间点对点的视频传输, 无法实现多客户间的多对多模式, Vert.x 通过事件总线技术既可以实现点对点模式, 更可容易实现群发模式的视频传输.

Vert.x 服务器实例运行时, 自动识别 CPU 的个数和内核数, 创建与 CPU 内核数对应的分布式实例, 将功能组件 Verticle 自动分布在每个服务器实例中, 并且内置动态负载均衡技术, 自动定位空闲的 Verticle 组件对象接收总线上到达的事件, 从而实现 Vert.x 的高性能高并发连接请求处理, 根据 Vert.x 社区测算, 一个单 Intel 至强 CPU 8 核的服务器上可以高效处理 40 万个链接请求, 这是传统的服务器技术无法实现, 如 WebLogic, JBoss,甚至 Node.js[5].

1.2 HTML5

本系统完全采用 Web 工作模式, 采用 HTML5 和 JavaScript 编程, 实现手机的跨平台应用, 可以运行在所有主流手机上, 克服了原生应用只能在单一平台运行的弊病. 手机视频的获取使用 HTML5 的 MediaStream API, 在 HTML5 页面内直接获得手机摄像头的实时视频. 通过 HTML5 的 Canvas 技术截取视频的二进制数据, 并编码为 Base64 字符类型, 使视频数据能在 Vert.x 的事件总线上传输. 视频接收端接收到事件总线上视频数据, 直接使用 HTML 的<img>标记进行显示, 当每秒接收超过 60 帧时, 人的眼睛无法识别出断续的画面, 显示为连续的实时视频.

2 系统设计与实现

2.1 系统总体架构设计

为保证视频监控的性能, 整个系统的服务端完全采用 Vert.x 实现[6], 包括 Web 服务器. 虽然 Vert.x 支持多语言开发, 但 JavaScript 是异步编程的最佳选择, 由于其专用的回调函数和事件处理机制, 与 Vert.x 的异步工作模式不谋而合, 本系统服务端和客户端都采用 JavaScript 语言实现编程.

服务器端 Vert.x 实例启动后, 自动创建事件总线对象, 所有 Verticle 通过编程取得事件总线对象实例后, 即可实现相互间的通讯和调用. 采用 Vert.x 的 Web 服务器模块实现了 Web 服务器, 保存客户端访问的所有 HTML5 页面和 CSS, JavaScirpt 脚本文件. 使用 MySQL5 数据库保存用户注册和登录信息, 通过专门的 Vert.x MySQL 模块实现数据库的操作, 另外使用专门的会话模块管理用户的会话数据.

客户端采用 HTML5, CSS, JavaScript, jQuery 和 Bootstrap 技术和框架, 尤其使用 Bootstrap3 框架设计的 Web 页面能自动适应手机, 平板, PC 的显示屏幕尺寸.

客户端使用 Vert.x 的 Event Bus Bridge(事件总线桥)和 SockJ 框架使得客户端连接服务器端的事件总线, 实现客户端与服务器, 以及客户端之间的文本数据和监控视频数据的传输.

整个系统的总体结构参见图 1 所示.

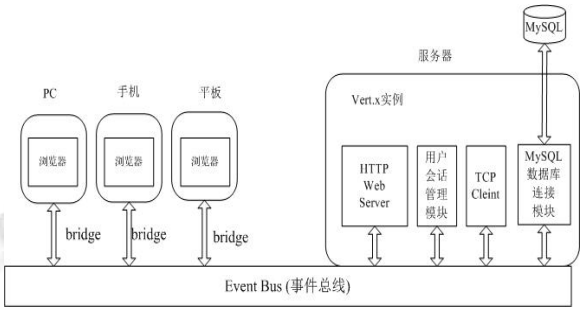

图 1　视频监控系统总体架构

2.2 Vert.x Web 服务器实现

为适应 Web 视频监控高性能要求, Web 服务器采用了专门的 Vert.x 模块 web–server~2.0.0, 其 Verticle 组件 WebServer.js 编程代码如下:

```
var container = require("vertx/container");
container.deployModule("io.vertx~mod–web–server~2.0.0–final", {
port: 8800, //设置服务的端口
host: "210.30.108.30",//设置服务器 IP 地址
bridge: true,
```

```
inbound_permitted: [ //设置允许的进入事件
{ address: 'city.oa.meeting.business.user'},
{ address: 'city.oa.meeting.business.meetinguser' },
{ address: 'city.oa.meeting.videoshow'}  ],
outbound_permitted: [//设置允许的输出事件
{ address: 'city.oa.meeting.business.user.add'},
{ address: 'city.oa.meeting.dao.user'},
{ address: 'city.oa.meeting.db'}]
});
```

程序中首先取得 Vert.x 服务器实例, 再通过 deployModule 方法载入 Web 服务器模块, 并配置 IP, 端口, 是否启用事件总线桥, 以及允许客户端在事件总线上发送和接收事件的地址. 当启用事件总线桥后, 浏览器客户端就可以连接到服务器实例中的事件总线, 进而实现数据的发送和接收

2.3 MySQL 数据库连接实现

系统总采用 MySQL 数据库保存各种数据, 包括登录用户, 用户保存的监控视频截取图片等. 由于 Vert.x 使用非阻塞异步工作模式, 不能使用像传统的 Java JDBC 那种阻塞同步方式操作数据库, 因此采用 Vert.x 提供的 MySQL 连接模块[7]. 该功能 Verticle MySQL.js 的代码示意如下:

```
var container = require('vertx/container');
var config =
{
        "address" : "city.oa.meeting.db",
        "connection" : "mysql",
        "host" : "210.30.108.30",
        "port" : 3306,
        "maxPoolSize":20,
        "username" : "root",
        "password" : "XXXXXXXYY",
        "database" : "cityvideosys"
};
container.deployModule("io.vertx~mod–mysql–postgresql_2.10~0.3.1",config);
```

在载入 MySQL 模块时, 先配置数据库连接信息, 其内部使用连接池模式以提高系统性能. 通过配置项 address 实现在事件总线上对数据库操作事件的监听, 实现对数据表的增删改查操作. 此数据库模块接收标准格式的事件, 以异步模式实现对数据库的操作, 如增加用户的事件 JSON 数据格式如下:

```
{
   "action" : "insert",
   "table" : "OA_USER",
   "fields" : ["userid", "password", "username", "ip"],
   "values" : [
      ["9001", "9001", "吕海东","172.30.39.240"]
   ]
}
```

2.4 数据访问层(DAO)和业务层(BO)实现

为实现组件间的解耦设计, 系统采用了与传统企业级应用相同的架构, 设计了数据访问层(Data Access Object–DAO)和业务处理层(Business Object–BO).

DAO 层组件专门负责数据的 CURD(增改查删)操作, 通过在事件总线上发送标准的指令和 SQL 语句实现对数据表的 insert, update, delete 和 select,并接收查询结果记录. 所有的参数和返回都使用 JSON 数据格式. 系统中将每个表都操作实现一个 DAO 层 Verticle, 将所有的这些 Verticle 封装到 DAO 层模块中. 其中对用户表的 DAO 层程序 UserDao.js 的调用 MySQL 数据库模块实现取得用户列表功能的简要实现代码如下:

```
var container = require('vertx/container');
var eventBus = require('vertx/event_bus');
//定义用户 DAO 接收事件
eventBus.registerHandler("city.oa.meeting.dao.user",function(args,responder){
    var action=args.action;
    var data=args.data;
    var sqlaction=null;
//DAO 层取得用户列表功能
if(action=="list"){
  sqlaction={   "action" : "select",
  "table" : "oa_user",
 "fields" : ["userid","userpassword", "username","ip"]
  };
}
if(sqlaction!=null)  {
//发送事件到数据库模块监听地址, 执行 SQL 操作
eventBus.send("city.oa.meeting.db",sqlaction,function(result)
{responder(result);//返回执行结果 });
```

```
});
```

DAO 模块使用事件总线的 registerHandler 方法, 注册 DAO 的事件处理, 并发送事件到数据库模块监听地址, 完成数据库操作. 数据库模块识别标准的事件参数, 通过 action 指定操作的类型, 当 action 为 prepared 指定采用预编译方式, 再通过 statement 指定要执行的 SQL 语句.

BO 模块[8]模拟实际的业务方法, BO 层 Verticle 注册此层的事件监听处理, 并解析收到的事件, 根据操作类型发送事件到 DAO 层地址. 客户端浏览器通过事件总线桥直接发送事件指令给服务器的 BO 层 Verticle, 以实现系统的业务功能. 如下是用户增加 BO 层组件 userAdd.js 的示意代码:

```
var container = require('vertx/container');
var eventBus = require('vertx/event_bus');
//定义事件总线上的增加管理员的 BO 层地址
eventBus.registerHandler("city.oa.meeting.business.user.
add",function(args,responder){
 var data=args;
 var config={
 action:"save",
 data:data
 };
//发送消息到 DAO 层监听处理器
eventBus.send("city.oa.meeting.dao.user",config,function
(result,info){  responder(result);  });
```

BO 层 Verticle 组装 DAO 层需要的 JSON 数据对象, 在事件总线上发送事件到 DAO 层的监听地址, 完成业务方法处理, 整个 Vert.x 应用中事件总线是关键, 是系统的中枢.

2.5 客户端视频读取和传输实现

客户端的核心是使用 HTML5 的 MediaStream API 提供的 getUserMedia 方法获取手机摄像头的视频, 示意代码如下:

```
videolocal=$("#videolocal")[0];
navigator.getUserMedia(constraints,function(stream){  i
f(window.URL){  videolocal.src=window.URL.create
ObjectURL(stream);  }
else{
    videolocal.src=stream;
}
```

使用 jQuery 框架获得页面的<video>对象, 将视频流引入到<video>标记中, 实现本地视频的实时显示. 获得视频流后, 再获取页面中的<canvas>对象, 使用 JavaScript 的定时机制, 对视频流进行拍照, 并发送到 canvas 对象. 使用 canvas 提供的获得图像数据方法 toDataURL()得到图像的 base64 字符编码, 再将用户的 ID 和图像编码以群分方式发送到事件总线, 实现监控视频群发的示意代码如下:

```
canvas=$("#canvas")[0];
context = canvas.getContext("2d");
var image=$("#videoimage")[0];
    //使用定时器实现每秒 60 帧的视频传输
    setInterval(function(){
    context.drawImage(videolocal, 0, 0, 100, 100);
    var dataurl=canvas.toDataURL();//取得视频数据
    //封装用户 ID 和视频数据
    var videodata={
      userid:userid,
      dataurl:dataurl
    };
    //发送图像数据
    eb.publish("city.oa.meeting.videoshow",videodata);
},1000/60);
```

Vert.x 的事件总线提供了 publish 方法实现发布/订阅模式的消息机制, 无论服务器还是浏览器客户在事件总线地址 city.oa.meeting.videoshow 上注册事件监听器均可接收到视频数据. 浏览器客户端使用 Event Bus Bridge 和 SockJS 连接到 Vert.x 服务器上的事件总线, 连接代码如下:

```
var eb = new vertx.EventBus(http://210.30.108.30:8800/
eventbus'); //Web 浏览器客户端连接到事件总线上
eb.onopen=function()
{
   //连接成功的回调处理, 所有处理放在此
}
```

在客户端连接 Event Bus 之前需要导入 Vert.x 的事件总线桥和 SockJ 类库.

```
<script src="sockjs/sockjs.js"></script>
<script src="vertx/vertxbus–2.1.js"></script>
```

为接收其他客户端发送的视频数据, 所有客户端均注册视频数据接收处理监听器, 其示意代码如下:

```
eb.registerHandler("city.oa.meeting.videoshow",function(
args,responder){
    var videouserid=args.userid;
    var dataurl=args.dataurl;
    var videoimage=$("#videotu"+videouserid)[0];
        videoimage.src=dataurl;
});
```

接收事件监听器接收到数据后，解析出发送者的 ID 以及视频数据，通过 ID 定位用户的图片元素<img>，将视频数据载入到图片标记对象，实现视频显示. 客户端在准备接收视频数据之前，向数据库发送请求在线用户列表事件，取得已经登录在线的用户列表，再根据返回的用户列表，使用 jquery 动态生成每个用户的视频显示图像标记,每个图像标记的 id 使用用户 ID 进行标示，用于此用户的视频显示，示意代码如下:

```
$("#meetingcontent").html("");
eb.send("city.oa.meeting.business.meetinguser.userlist",d
ata,function(result){
    var users=result.results;
    for(var i=0;i<users.length;i++){
                $("#meetingcontent").append("<div
class='video'><div class='video'> <img
id='videotu"+users[i][0]+"'/>
</div><div>"+users[i][1]+"</div></div>"); }
});
```

图 2 为模拟多个用户登录情况下的实时视频监控的显示 Web 页面. 经过实际测试，如果使用 WiFi 和 4G 联网，效果非常好，而使用 3G 则出现不连贯现象，发生丢帧现象，因为网速慢时，事件在事件总线上堆积，无法及时得到接收处理，Vert.x 会自动舍弃超时没有被接收的事件，以便保持事件总线畅通.

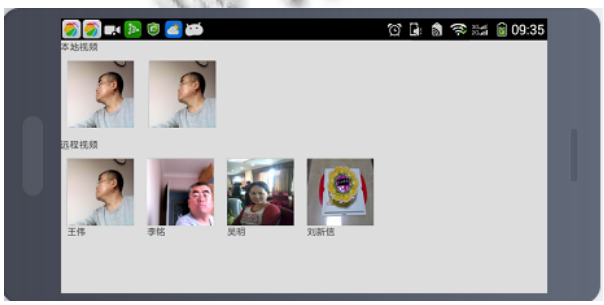

图 2 使用 Android 手机的视频监控页面

3 结语

未来移动互联网将进一步发展和普及，随着 4G 和宽度速度的大幅度提高，Web 将从文本应用向视频和声音多媒体应用方向发展，进而对服务器的性能有了更高的要求. 如何在现有的硬件投资下，通过软件技术来改进移动应用的性能，Vert.x 给出了最佳的解决方案. 采用 Vert.x 全新的工作模式，利用其分布式的事件总线机制，可以开发出常规技术无法实现的创新性的应用，以适应未来以移动终端为主的大数据高并发的实时性应用系统. 本文的移动实时视频监控系统，如果使用传统的技术如 WebRTC 或流媒体服务器技术都难以高效实现. 未来以 Vert.x 和 Node.js 为代表的单线程、异步模式的服务器技术必将取代现有的多线程、同步模式的技术.

参考文献

1 Vert.x 官方网址.http://vertx.io/.

2 Node.js 官方网址.http://www.nodejs.org/.

3 Parviainen T. Real-time Web Application Development using Vert.x 2.0. Birmingham-Mumbai: Packt Publishing, 2013.

4 Enterprise messaging and integration with Vert.x. http://www.javaworld.com/article/2078838/mobile-java/open-source-java-projects-vert-x.html/.

5 Vert.x 与 Node.js 性能比较测评. http://www.cubrid.org/blog/dev-platform/inside-vertx-comparisonwithnodejs/.

6 Scarduzio S. Instant Vert.x. BIRMINGHAM-MUMBAI: Packt Publishing, 2013.

7 Vert.x MySQL 模块.https://github.com/ vert-x/mod-mysql-postgresql.

8 Vert.x 会话管理模块.https://github.com/campudus/vertx-session-manager.