计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(3):127−131 [doi: 10.15888/j.cnki.csa.007304]

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

基于 Websocket 的车联网报警推送系统①

李先懿, 郭正光

(上海势航网络科技有限公司, 上海 201702)
通讯作者: 李先懿, E-mail: lixianyi@live.com

摘　要: 报警处理是车联网系统中的重要功能. 传统车联网系统一般采用 AJAX 轮询的方式定时拉取报警信息, 既不能保证报警实时性和准确性, 也给服务器造成压力. 本文设计了一种基于 Websocket, Redis 等技术的报警实时推送系统, 并对推送性能做了测试和分析. 结果表明: 使用 Websocket 推送方式处理报警信息, 既提高了报警的实时性和准确性, 又提高了推送的吞吐量, 降低网络流量.

关键词: Websocket; Redis; 车联网; 报警推送; 实时性

引用格式: 李先懿,郭正光.基于 Websocket 的车联网报警推送系统.计算机系统应用,2020,29(3):127–131. http://www.c-s-a.org.cn/1003-3254/7304.html

Telematics Alarm Pushing System Based on Websocket Protocal

LI Xian-Yi, GUO Zheng-Guang

(Shanghai Shihang Network Technology Co. Ltd., Shanghai 201702, China)

Abstract: Alarm processing is fundamental function in Telematics System. AJAX pulling is common method in traditional system. But AJAX pulling is not real-time communication, and the server suffers unnecessary requests. In this study, we design a telematics alarm pushing system based on Websocket and Redis. Then, we test this system. The results show that Websocket is an ideal technology for alarm pushing. It improves network throughput, reduces latency and network traffic.

Key words: Websocket; Redis; telematics system; alarm pushing; real-time

1 概述

近年来, 随着电子信息技术、移动通信技术、物联网技术的快速发展, 车联网技术也有长足的进步[1,2]. 与此同时, 物流运输企业和政府监管部门都对车辆安全更加重视. 这也推动了新技术在车联网中的应用. 在车联网系统中, 报警通知和处理是非常重要的功能模块. 按照交通部相关法规及各省交管部门的要求[3], 当车辆发生超速、疲劳驾驶等危险驾驶行为时, 交通部联网联控平台的车联网运营商需要及时收到通知, 针对车辆异常行为进行相应处理. 因此, 报警通知显得尤为重要.

当车辆发生异常报警时, 终端通过 socket 实时上报到车联网服务器. 服务器需要将报警信息实时通知到用户. 传统 BS 架构的车联网系统中, 一般通过 AJAX 轮询[4]的方式拉取报警信息. 这种方式实现简单, 但它的弊端也是显而易见: (1) 轮询是有时间间隔的, 所以报警是有延迟的. 过短的轮询间隔产生大量的无效请求并增大服务器压力. (2) 当用户数量增多时, 轮询会产生大量的网络流量, 增加不必要的带宽损耗, 并导致服务器产生很大并发压力. (3) 当终端数据上报频率高于 AJAX 轮询频率时, Web 客户端还没来得及做一次 AJAX 请求, 新的报警数据已经覆盖了旧的报警

① 收稿时间: 2019-07-26; 修改时间: 2019-08-29; 采用时间: 2019-09-10; csa 在线出版时间: 2020-02-28

数据, 会造成报警数据丢失. 因此, 需要采用新的技术手段解决车联网系统中海量报警数据推送问题.

近年来, 由于 HTML5 技术的广泛普及, 主流浏览器对 HTML5 标准支持更加完善, 客户端和服务器之间的实时数据交换变得更加容易. Websocket 是 HTML5 新标准中的通信机制[5,6], 能够实现稳定全双工实时通信, 具有简洁高效的特点. 在 Websocket API 中, 浏览器和服务器只需要完成一次握手, 两者之间就可以直接创建持久性的连接, 并进行双向数据传输, 占用的网络带宽较少. 它不需要安装浏览器插件, 所以跨浏览器兼容性更好. 因此, Websocket 技术可作为车联网系统中报警推送的理想方案.

2 系统设计

针对上述 AJAX 问题并考虑到 Websocket 的优点, 我们设计一套基于 Websocket 技术的报警推送系统. 该系统主要包含 4 个模块: 网关、Redis 数据库、Websocket 推送服务、Websocket 客户端及前端页面. 整个系统的架构如图 1 所示.

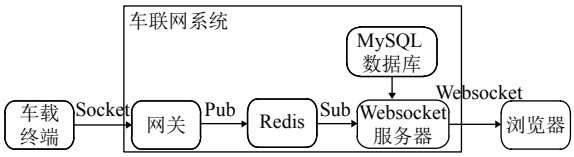

图 1 Websocket 报警推送系统架构

在车联网系统中, 终端通过接入网络[7](包括 2G/3G/4G 等无线移动通信网络或 WLAN 等网络) 按照《道路运输车辆卫星定位系统北斗兼容车载终端通讯协议技术规范》(简称 JT/T 808) 协议[8]上报定位数据、报警数据和其它附加数据. 网关模块负责维持与众多终端的 socket 长连接, 接收终端上报的数据, 解析数据并写入数据库. 根据上报数据的类型不同, 一些数据写入关系型数据库以便长期持久化. 这些历史数据将用于车辆轨迹查询、行程分析、报警统计等功能. 另外一些需要实时查询和推送的数据写入 Redis 缓存数据库. 网关通过两种方式写入 Redis: 通过 set/hset 方式写入 Redis 以便其它模块查询实时数据; 通过发布模式 (publish) 写入 Redis 以便实时通知其它模块.

Redis 是一个开源的内存数据结构存储系统, 它可以用作数据库、缓存和消息中间件[9]. Redis 的所有数据均位于服务器的主内存中, 因此读写速度非常快. 每秒可以执行 10 万次以上的写入操作. Redis 的发布订阅模式实现了一个简单的消息队列的功能. 引入 Redis 的目的在于: (1) 降低网关与后续 Websocket 推送服务间的耦合, 从而保障网关的稳定性. 即使网络异常导致 Websocket 推送服务产生积压或阻塞, 也不会影响网关与终端的正常通信, 不会影响数据写入关系型数据库. (2) 当多个程序订阅 Redis 消息时, 数据可以在系统内不同模块间实时分发, 实现数据同步. (3) Redis 本身也承担缓存的作用, 可以直接在 Redis 里查询所有车辆的最新一次位置信息、报警状态.

Websocket 推送服务主要承担权限判断、Websocket 推送、推送统计等功能. 它从 Redis 订阅了报警频道 (channel). 当有报警信息从 Redis 推送过来, 它提取报警内容以及该报警对应的车辆信息、终端信息, 然后从数据库查询用户与车辆的对应关系, 判断是否需要将本条报警信息推送给用户. 由此可以做到报警推送与用户权限绑定, 具有查看车辆权限的用户才会收到报警推送. 另外, 可也根据用户喜好设置, 过滤特定类型的报警推送, 只推送用户关注的报警类型.

Web 前端采用 Socket.io 的 js 库. Socket.io 是一个跨浏览器、支持 Websocket 实时通讯的 js 库[10]. 它支持实时、双向、基于事件的数据通信, 可在不同平台、浏览器、设备上工作. Socket.io 除了支持 Websocket 通讯协议外, 还支持许多种轮询 (Polling) 机制以及其它实时通信方式, 并封装成了通用的接口. Socket.io 能够根据浏览器对通讯机制的支持情况自动从 Websocket, Adobe Flash Socket, AJAX long pulling, AJAX multipart streaming, Forever IFrame, JSONP polling 中选择最佳的方式来实现网络实时应用. 这样前端开发 Websocket 变得更简单.

3 系统实现

整个车联网系统的网关和后端使用 Java 语言开发, Web 前端基于 vue.js 框架开发.

车联网平台接入的终端较多, 在线的终端可能数以万计. 这些终端通过 TCP 长链接与平台通信. 因此需要一个性能强大的网络连接库. 本系统网关部分采用 Netty 库处理 socket 连接和数据收发. Netty 是一个基于 Java NIO 的客户端/服务器网络应用框架[11]. 它隐藏背后复杂的网络操作, 提供一个易于使用的 API 框架.

Netty 框架的一大亮点是基于 EventLoop 的高效线程模型. 它是一个高性能、异步事件驱动的 NIO 框架, 所有 IO 操作都是异步非阻塞的. 高性能的 Netty 可以处理数万终端的连接和数据收发.

Netty 通过 bind() 方法监听 9300 端口, 等待终端连接. 当终端上报数据时, 继承自 ChannelInboundHandlerAdapter 的 GatewayHandler 类的 channelRead 方法接收数据并解析报警. 然后通过 jedis 的 publish 方法发布报警到 Redis 里面. 核心代码如下:

```
ServerBootstrap b = new ServerBootstrap();
b. group(bossGroup, workerGroup)
. channel(NioServerSocketChannel. class)
. childHandler(new ChannelInitializer<SocketChannel>() {
@Override
public void initChannel(SocketChannel ch) throws Exception {
        ch. pipeline(). addLast(new GatewayHandler());
}
});
//网关监听 9300 端口
ChannelFuture f=b. bind(9300). sync();
f. channel(). closeFuture(). sync();
…
//网关将报警发布到 redis
jedis. publish("channel _alarm", alarm);
```

在本系统中, Redis 既用作了缓存数据库, 也充当了消息队列. Redis 服务器采用了 6 个 Redis 节点集群的方式部署, 包括 3 个主节点, 3 个从节点. 这样可以做到自动分配数据到不同的节点上, 提高了性能. 部分节点失效的情况下还能够继续提供服务, 提高了系统的高可用性.

Websocket 推送服务采用开源 netty-socketio 方案 (https://github.com/mrniko/netty-socketio). 它是 Socket. io 服务器端的一个基于 Netty 框架的 Java 实现. 它功能非常强大, 简单易用, 稳定可靠.

Netty-socketio 推送服务的 ConnectListener 监听了 Websocket 连接事件. 当用户通过浏览器登录本系统, 服务端会收到一个唯一标识 (UUID). Clients 集合保存了所有已登录用户的标识. 当终端上报一条报警数据后, 程序从数据库查询该报警对应的车辆, 再根据车辆和用户的关系, 查找到该报警需要通知的一个或多个用户. 如果用户的个人设置里配置了接收报警推送, 则从 clients 集合里查找到该用户对应的 Websocket 连接, 将该报警通过此连接推送到 web 前端. Websocket 推送服务核心代码如下:

```
//保存所有已登录用户建立的 Websocket 连接标识
List<UUID> clients=new ArrayList<UUID>();
//配置 websocket
Configuration conf=new Configuration();
conf. setPort(9092);
conf. setBossThreads(128);
conf. setWorkerThreads(128);
conf. setUseLinuxNativeEpoll(true); //epoll 性能更高
SocketIOServer server=new SocketIOServer(conf);
ConnectListener listener=new ConnectListener() {
    @Override
    public void onConnect(SocketIOClient client) {
    //用户登录系统, 建立 websocket 连接
    clients. add(client. getSessionId());
    }
};
server. addConnectListener(listener);
server. start();
…
//向 Web 前端推送报警
socket. sendEvent(topic, alarm);
```

前端引入 socket.io.js 监听 Websocket 事件, 例如建立连接、断开连接、收到数据等事件[12]. 当收到报警时, js 解析报警内容, 将结果展示在 web 界面. 核心代码如下所示:

```
// 建立连接
var socket=io. connect('http://192.168.0.2:9092');
// 监听 message
socket. on('message', function (data) {
    var html=document. createElement('p')
    html. innerHTML='车牌为: '+data. vehicle+'的车辆发生了类型为'+data.alarm+'的报警, 请及时处理';
    document. getElementById('content'). appendChild(html);
});
```

整个前端页面包含了一套完整的车联网平台, 包

括车辆监控、轨迹回放、报警处理、指令下发、报表查询等功能. 报警推送只是其中一个模块. 当前端收到报警推送时, 通过弹框和播放声音等方式明显提示用户. 车辆监控、报警处理、报表查询等功能也会同步刷新报警数据.

4 系统测试与分析

系统测试环境如下: 服务器硬件配置为 16 核, 8 GB 内存. 服务器操作系统为 Centos 7. Java 使用 Oracle JDK1.8.0_171. Web 服务器使用 Tomcat 8.5.32. Redis 版本为 4.0.10. Netty-socketio 版本为 1.7.13. 客户端环境为 Windows 10 操作系统, 浏览器为支持 Websocket 的 Google Chrome 67.0.3396.99.

我们定义报警的延迟为: 从终端产生报警的时间到用户浏览器收到报警的时间差. 严格测试报警延迟, 需要先将终端和用户电脑校时. 这里我们使用模拟终端巧妙回避校时的问题. 模拟终端是一个可运行在 Windows 上的程序. 它模拟了车载终端的数据采集、网络数据收发逻辑. 将模拟终端和浏览器运行在同一台 Windows 电脑上, 比较终端产生报警的时间戳和浏览接收到报警的时间戳, 即可计算出报警推送的延迟.

使用图 2 所示模拟终端发送测试数据. 通过模拟终端数据发送时间和 Chrome 开发者工具控制台日志输出时间可以计算报警延迟. 每秒推送一次报警, 共推送 100 条报警数据, 从终端上传报警数据到浏览器收到报警推送的时间延迟平均值为 8 ms. 传统 AJAX 轮询间隔大约是 10 到 30 s. 可见报警实时性大大提高.

图 2 模拟终端报警配置界面

采用开源工具 apache jmeter 对 netty-socketio 服务器作了压力测试. Jmeter 的 Websocket Sampler 插件可以模拟 Websocket 并发连接. 在每个并发连接数下, 推送 100 条数据, 取推送延迟的平均值. 测试结果如图 3 所示. 结果表明, 随着 Websocket 并发连接数增长, 推送延迟并未出现大幅度增长, 而是在 7.5 ms 左右波动. 这也反映出 netty-socketio 的高并发、低延迟特性.

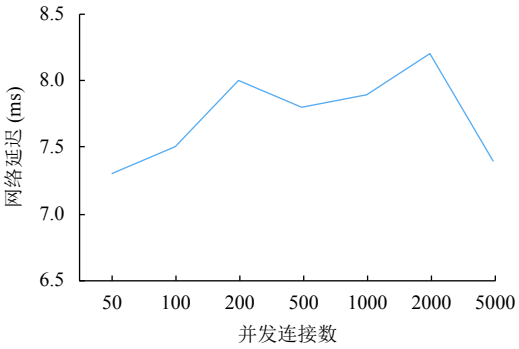

图 3 Websocket 推送延迟与并发连接数曲线

把模拟终端的数据上报时间间隔设定成 1 s, 连续发送 1000 条报警数据. 浏览器控制台输出数据表明接收到 1000 条报警. 这说明 Websocket 并未丢失任何一条数据, 不会出现 AJAX 方式的报警覆盖问题, 报警推送的准确性得到保证.

通过 Chrome 浏览器开发者工具查看网络连接, 当页面加载时, 服务响应的 HTTP 头信息中包含了这两项: Connection: Upgrade 和 Upgrade: Websocket, 可知客户端和服务器在握手期间从 HTTP 协议升级为 Websocket 协议. 之后 Websocket 一直保持长连接. 而 AJAX 轮询时, 每次建立 HTTP 连接, 都要发送 HTTP 报头等信息. 通过 wireshark 抓包软件可知, 通常一次 HTTP 请求和响应, 需要发送大约 570 B 的 HTTP 头部信息[5]. 这个 HTTP 报头甚至比消息本身还要大. 而使用 Websocket 的额外开销只有 2 B, 建立连接之后便可以二进制帧格式传输纯数据. 我们假定一次报警的数据为 50 B. 通过 AJAX 方式拉取 1000 条报警生产的网络流量为: (570+50)×1000=620 000 B. 而通过 Websocket 传输的网络流量为 (2+50)×1000=52 000 B. AJAX 方式产生的网络流量是 Websocket 网络流量的 11 倍. 当报警推送量大后, Websocket 比 AJAX 轮询节省大量的网络流量.

由于主流的浏览器已经支持 Websocket, 前端

socket.io.js 也具有较好的自适应性, 所以整个系统具有较好的浏览器兼容性. 经过测试, 报警推送功能在最新版的 Chrome, Firefox, IE11 等常用浏览器上均能正常工作并具有较好的性能.

本系统已经在生产环境中使用. 该平台包含 4.5 万台入网终端, 其中大约 2.1 万在线终端. 日均产生报警数据 750 万条左右. 平均每秒钟推送 86 条报警. 服务器的 CPU、内存和网络均保持较低压力状态. 客户端 Chrome 浏览器的 CPU 和内存占用保持平稳, 说明浏览器能承受如此大量的报警推送. 系统已经稳定运行数月, 证明这是一套可用于生产环境的稳定系统.

5 结束语

根据车联网的特定业务场景, 设计了一种基于 Websocket 技术的车联网报警推送系统. 对比 AJAX 轮询技术和 Websocket 技术可知, 使用 Websocket 方案以后, 大大降低报警的推送延迟, 提高了报警推送的吞吐量, 保证报警信息及时准确地推送到客户. 随着 HTML5 技术的普及, Websocket 将逐渐成为 Web 实时通信的主流技术.

本文介绍的方案解决了报警实时性和准确性等问题. 但是当数万终端每天产生 750 万报警数据时, 如何让用户方便查看并处理海量报警成了新的问题. 报警聚合和报警联动将是后续研究的方向.

参考文献

1 程刚, 郭达. 车联网现状与发展研究. 移动通信, 2011, 35(17): 23–26. [doi: 10.3969/j.issn.1006-1010.2011.17.004]

2 王建强, 吴辰文, 李晓军. 车联网架构与关键技术研究. 微计算机信息, 2011, 27(4): 156–158, 130. [doi: 10.3969/j.issn.2095-6835.2011.04.064]

3 中华人民共和国交通运输部. JT/T 796-2011 道路运输车辆卫星定位系统 平台技术要求. 北京: 人民交通出版社, 2011.

4 孙清国, 朱玮, 刘华军, 等. Web 应用中的服务器推送技术研究综述. 计算机系统应用, 2008, 17(11): 116–120. [doi: 10.3969/j.issn.1003-3254.2008.11.030]

5 李代立, 陈榕. WebSocket 在 Web 实时通信领域的研究. 电脑知识与技术, 2010, 6(28): 7923–7925, 7935.

6 薛陇彬, 刘钊远. 基于 WebSocket 的网络实时通信. 计算机与数字工程, 2014, 42(3): 478–481. [doi: 10.3969/j.issn.1672-9722.2014.03.030]

7 任开明, 李纪舟, 刘玲艳, 等. 车联网通信技术发展现状及趋势研究. 通信技术, 2015, 48(5): 507–513. [doi: 10.3969/j.issn.1002-0802.2015.05.001]

8 中华人民共和国交通运输部. JT/T 808-2011 道路运输车辆卫星定位系统 终端通讯协议及数据格式. 北京: 人民交通出版社, 2011.

9 曾超宇, 李金香. Redis 在高速缓存系统中的应用. 微型机与应用, 2013, 32(12): 11–13. [doi: 10.3969/j.issn.1674-7720.2013.12.004]

10 黄经赢. 基于 Socket. io+Node. js+Redis 构建高效即时通讯系统. 现代计算机, 2014, (13): 62–64, 69. [doi: 10.3969/j.issn.1007-1423(z).2014.13.016]

11 金志国, 李炜. 基于 Netty 的 HTTP 客户端的设计与实现. 电信工程技术与标准化, 2014, (1): 84–88. [doi: 10.3969/j.issn.1008-5599.2014.01.027]

12 王佃来, 宿爱霞, 安晏辉. 基于 WebSocket 的消息推送系统. 计算机系统应用, 2017, 26(9): 87–92. [doi: 10.15888/j.cnki.csa.005973]