面向泵站远程监控的 Push 技术设计

周建群,李千目
(南京理工大学 计算机科学与工程学院,江苏 南京 210094)

摘 要:B/S 架构可以直观地展示泵站内设备运行状态的相关数据,大大提高了系统管理水平和运行人员的工作效率。数据实时推送成为了泵站远程监控的一项重要技术。通过对现有的实时 Web 应用技术进行研究,发现 Ajax 轮询会造成系统消息延迟,大量使用会增加服务器和客户端间的通信开销。基于长连接的 Comet 技术可以实现服务器 push,但是单台服务器只能维护少量长连接请求,不容易扩展。针对这种情况,提出了一种基于 WebSocket 协议的实时 Web 应用技术方案,并将其应用于泵站远程监控系统中。实验结果表明,该方案能够有效降低通信开销,提升服务器性能。

关键词:泵站远程监控;Ajax 轮询;WebSocket;服务器推送
中图分类号:TP39 **文献标识码**:A **文章编号**:1673-629X(2015)09-0164-05
doi:10.3969/j.issn.1673-629X.2015.09.035

Design of Push Technology for Remote Monitoring of Pumping Station

ZHOU Jian-qun, LI Qian-mu
(School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China)

Abstract: The B/S architecture can intuitively display related data of the running state of the equipment in pumping station, which not only improves the quality of system management, but also rise the efficiency of personnel working. Real-time data publication has become a key technology in the remote monitoring of pumping station. The study on the existing technology of real-time Web applications found that Ajax polling could cause the system message delay, and could increase the overhead of communicating between the server and the client by heavy use. Comet based on long connection can solve server push, but a single server can only maintain a few long connection request, and it's not easy to extend. In view of this, propose a real-time Web application technology scheme based on the WebSocket protocol, which has been used to the remote monitoring of pumping station. Experimental results show that this scheme can effectively reduce the overhead of communicating between the server and the client, and increase the system performance.

Key words: remote monitoring of pumping station; Ajax polling; WebSocket; server push

0 引 言

泵站远程监控是指利用计算机通过网络实现对泵站内设备运行状态和相关数据的监视和控制。需要监控的数据主要包括电压、电流、温度、水位和流量。B/S 架构可以直观地在浏览器上反映泵站设备的实时监测数据。为了便于工作人员更有效地进行管理,更具有针对性地解决出现的问题,能够直观地显示系统中各类数据的可视化技术应运而生。

数据实时推送已成为泵站远程监控的重要技术,也是管理人员最为关心的对象。通过对这些数据的监测和分析能够了解设备的性能、运行状态和健康状况,从而保证整个泵站的正常稳定运行。在汛期这些数据显得尤为重要,通过对某些重要数据的监控,可以实现泵站自动化控制。当汛期水位上升过快时,决策系统可以根据水位上升的幅度和速率来决定是否打开泵机进行排水,同时决定开启几台泵机。

早期 Web 实时通讯系统采用浏览器周期发送 Ajax 请求来得到服务器上的最新数据,这种方式不仅会造成消息延迟,而且过多的 Ajax 请求增加了服务器负荷。后来出现了 Comet 技术,它改变了 Ajax 轮询向服务器"pull"请求的方式,通过在浏览器和服务器之间维护一条 HTTP 长连接,服务器在一定时间内可以保持这个连接不被关闭。连接建立完成后,服务器端可以直接将新数据通过该连接推送到服务器端,仍然

收稿日期:2014-10-18 **修回日期**:2015-01-22 **网络出版时间**:2015-07-21
基金项目:国家自然科学基金资助项目(61272419);江苏省未来网络前瞻性研究项目(BY2013095-3-02)
作者简介:周建群(1989-),男,硕士研究生,研究方向为网络与信息安全;李千目,博士,教授,研究方向为网络与信息安全。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20150721.1453.103.html

是通过 HTTP 协议发送请求。为了弥补单一方案的不足,可以结合长轮询和轮询方案[1],在不同的使用场景下,灵活切换数据推送方式,这在满足实时性要求的同时兼顾了服务器资源。还有一种方案在服务器和浏览器之间的通信通过消息中间件来传输,消息中间件采用缓冲池技术[2]。这能有效提高服务器性能,但是所有的消息都通过消息中间件来传输,这对消息中间件的性能和带宽要求比较高,同时消息隔层传递,在大量并发的情况下,消息延迟会更加明显。

1　Web 实时通信方式

1.1　Ajax 轮询

Ajax(Asynchronous Javascript and XML)通过 HTTP 请求在后台与服务器进行少量数据交互,使浏览器可以实现异步更新,而不需要刷新整个网页。Ajax 轮询方式需要在浏览器端设置一个固定的"时钟周期"去轮询服务器,以获取服务器端最新的数据。但是"时钟周期"大小的设定是一个需要解决的问题。如果轮询的周期太长,浏览器将无法及时获取服务器端的最新数据,服务器和浏览器之间产生了消息延迟。如果轮询的周期过短,浏览器频繁发送 HTTP 请求,既会造成服务器端的资源浪费,也会产生冗余的请求,增加不必要的通信开销[3]。所以轮询周期只能根据实际应用的场景设定一个经验值。如图 1 所示,Ajax 轮询在请求频率较低的情况下,不能及时获取服务器端最新更新的数据[4],而这样的消息延迟在泵站汛期排水监测期间是不可容忍的。

轮询方法的主要缺点是:随着浏览器端访问用户数量的增加,服务器和浏览器之间的通信流量会呈爆发性增长。最坏的一种情况是对不频繁发生更新的应用程序使用轮询,在这种情况下,相当数量的客户端轮询是没有必要的,服务器对这些轮询的响应只会是"数据未发生变化"。

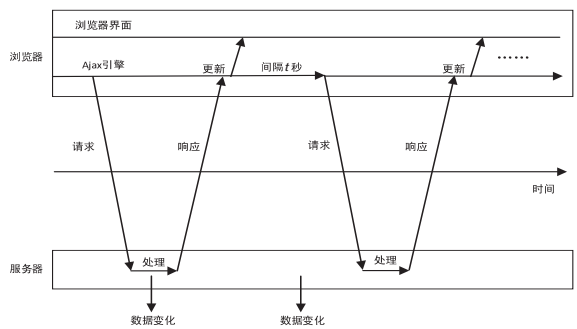

图 1　Ajax 轮询

1.2　基于长连接的 Comet 技术

基于 HTTP 长连接的技术又被称为 Comet。基于 HTTP 的 Web 应用服务器和浏览器的一次交互的主要步骤:

(1)浏览器连接到 Web 服务器;

(2)浏览器发送 HTTP 请求;

(3)Web 服务器返回响应结果;

(4)服务器断开连接。

由此可见,HTTP 请求是一个基于短连接的无状态请求。浏览器只有通过请求才能获取服务器的最新状态,服务器无法主动将信息发送给浏览器。Comet 打破了这种限制,在浏览器和服务器之间建立一条长连接,此后,服务器一直在这条长连接上发送、接收请求。Comet 有两种实现方式:一种是基于长轮询的方式,另外一种是基于 iframe 流的方式。长轮询方式是在连接建立完成后,服务器保持住这个连接,直到产生新的数据后,再将数据"push"到浏览器端。服务器可以复用这条连接一直发送数据。服务器需要一直维持这条长连接,每新增一个客户端,服务器就需要开辟一个线程来维护这条长连接,这对服务器性能要求会很高,系统的稳定性也无法得到保证。因此使用基于长轮询的长连接需要解决系统的稳定性和服务器推送数据的质量[5]。基于 iframe 流的长连接需要在页面嵌入一个隐藏的 iframe,服务器和浏览器的长连接就建立在这个 iframe,数据的传输直接通过 iframe 来完成,可以通过 JavaScript 完成数据的发送、接收和解析。iframe 流方式明显的优点就是可以兼容各大主流浏览器。但是,HTTP1.1 规定在同一个客户端能够维护的 iframe 流长连接不能超过两个,同一个客户端过多的长连接会被服务器阻塞[6],这样会影响用户体验,同时设计开发人员应该尽量避免在同一个浏览器上建立两条 iframe 长连接,这种限制对实时 Web 应用的扩展是非常不利的。图 2 为长连接的工作原理图。长连接建立完成后,服务器可以一直在该连接发送和接收数据,直到通信错误或者连接重建时才被动断开连接。

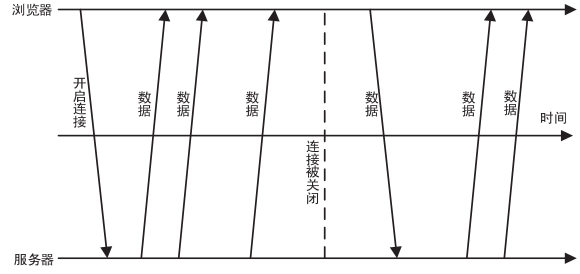

图 2　HTTP 长连接原理

DWR(Direct Web Remoting)和 Pushlet 都是开源的 Comet 框架。浏览器页面可以通过 DWR 直接调用 Web 服务器上的 Java 代码,建立 RPC 连接[7]。DWR2.0 新增了 server push 功能,直接通过异步传输将数据从 Web 服务器推送到浏览器。Pushlet 采用了发布/订阅的模式[8],服务器是数据的发布者,服务器端需要动态维护一个订阅客户端的事件队列,当有新

数据产生或者数据发生变化时,服务器会将数据推送到队列中的每一个客户端,服务器为每一个用户单独开辟线程来处理事件,这会增加服务器开销[9]。服务器还需要为每一个客户端产生一个不同的会话 ID,而这个会话 ID 和会话对象会以键值对的方式被保存到 HashMap 中,由服务器来维护。

Comet 的使用需要结合不同的使用场景来做针对性设计。在大型 Web 实时应用中,首先需要考虑的是服务器的稳定性和健壮性。如果采用基于 Comet 的长连接,服务器会为每一个客户端开辟线程建立长连接通信,而一台服务器的线程池中的线程数量是有限的,当线程数量达到上限时,会影响 Web 应用的其他功能,同时,新连接也会被阻塞[10]。此时,应该考虑在服务器端采用集群负载均衡技术,将长连接的请求分发到每一台服务器上,通过监测每台服务器的性能综合参数,动态地决定将请求分发到哪一台服务器上[11]。集群会显著增加服务器和客户端以及服务器之间的通信开销,并且需要动态维护各个服务器的性能参数,应用的场景也比较有限。

2 基于 WebSocket 的数据实时推送

2.1 WebSocket 介绍

浏览器端通过 HTTP 协议只能实现和服务器的单向通信,Comet 虽然可以一定程度上模拟双向通信,但是效率较低,并且需要服务器提供较好的支持。WebSocket 是 HTML5 提出的一个新的通信协议,它在服务器和浏览器之间架设了一个全双工(full-duplex)的通信通道[12],并使用套接字来传输数据。WebSocket 所建立的连接本质是一个 TCP 全双工通信连接,连接的双方能够自由进行数据传输,兼顾了传输量和服务器的稳定性。WebSocket 协议可使通信双方了解彼此的状态,而 HTTP 协议是无状态的协议,服务器端无法将自身的数据变化及时通知给客户端,浏览器端只能通过发送 HTTP 请求才能获取服务器上的最新数据。主流浏览器的最新版本都对 WebSocket 提供了良好的支持,JavaEE7 也在服务器端实现了 WebSocket 协议。

2.2 WebSocket 通信原理

在 WebSocket API 中,浏览器和服务器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。两者之间就直接进行数据互送了。WebSocket 建立连接时由浏览器发起建立连接的“握手”请求[13]。这个请求和一般的 HTTP 请求报文的区别主要在头信息,WebSocket 请求会在头消息上附加“Upgrade:WebSocket”,表明这个请求是基于 WebSocket 协议的。服务器解析这段头消息,并返回响应结果,连接建立完毕。连接的双方就可以在这个连接发送和接收数据。在数据传输完成后,服务器端和浏览器端均可以主动发起连接关闭请求。

WebSocket 在建立连接时,浏览器和服务器端需要有一个握手的过程。此过程完成后,浏览器和服务器之间就建立了一条类似 TCP 的全双工通道。使用 WebSocket 协议实现实时 Web 应用的优点是连接的双方相互沟通的消息头只有 2 字节,远比 HTTP 的头消息要小,通信的双方可以彼此推送消息,这是相对 HTTP 协议的一个重要革新。握手连接建立时,浏览器端发送请求的报文格式如下:

```
GET /demo HTTP/1.1
Host: localhost
Connection: Upgrade
Sec-WebSocket-Key2: 12998 5 Y3 1 .P00
Upgrade: WebSocket
Sec-WebSocket-Key1: 4@1 46546xW%0l 1 5
Origin: http://localhost
[8-byte security key]
```

服务器接收到浏览器端的请求后,返回给浏览器的报文格式如下:

```
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: http://localhost
WebSocket-Location: ws://localhost/demo
[16-byte hash response]
```

至此,WebSocket 连接已经成功建立。WebSocket 请求和 HTTP 请求报文主要区别在“Upgrade:WebSocket”,它表示这是一个 WebSocket 请求,这段报文表示该请求使用的是 WebSocket 协议,是 HTTP 的一个升级协议[14]。浏览器发送请求的信息里还有“Sec-WebSocket-Key1”、“Sec-WebSocket-Key2”和“[8-byte security key]”这样的头信息。这是浏览器提供给服务器的握手信息,握手信息包含了加密认证 key 参数,会和请求一起发送到服务器端。服务器端接收到请求后会解析这些头信息,并在握手的过程中依据这些信息生成一个 16 位的安全密钥并返回给浏览器,表示服务器收到了浏览器的握手请求,同意建立安全的 WebSocket 连接。WebSocket 的通信原理如图 3 所示。双

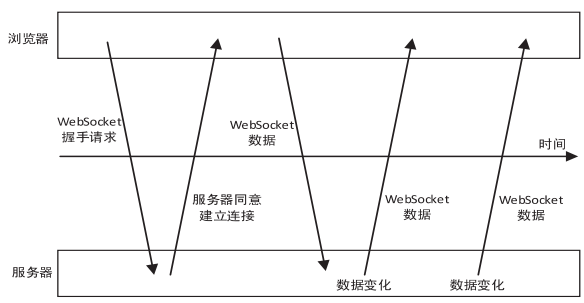

图 3 WebSocket 通信原理图

方连接建立完成后,采集到的新数据直接通过刚才创建的连接推送到浏览器。

3　实验结果及对比

3.1　网络负载流量

泵站远程监控主要监控的数据是电压、电流、温度、水位和流量。考虑到泵站是分散在不同的地域,采集系统每次需要采集多个泵站的实时数据,然后通过 Web 服务器推送到浏览器端,供用户查看。考虑这种使用场景:服务器每隔 5 s、10 s、30 s 采集一次数据,泵站采集的数据有电压、电流、温度、水位和流量。以采集五个泵站的数据为例,每个泵站有四台设备需要采集数据,产生的数据大小为 320 字节,按照 Ajax 轮询,轮询周期为 5 s,如果轮询时没有采集到新的数据,就直接把最近的数据返回给浏览器,HTTP 请求和响应 header 大小为 660 字节。使用 WebSocket 时,在连接建立完成后,通信开销是 2 字节。图 4 是 Ajax 轮询和 WebSocket 在上述使用场景下网络流量比较结果。

当使用 Ajax 轮询时,上述场景下的网络负载为:

5 s 产生一次数据:(660+320)×5×4 = 19 600 bytes = 19.1 kB;

10 s 产生一次数据:(660+320)×5×4×2 = 39 200 bytes = 38.2 kB。每两次轮询,有一次数据是未发生变化的;

30 s 产生一次数据:(660+320)×5×4×6 = 117 600 bytes = 114.8 kB。每六次轮询,只有一次轮询数据是发生改变的,83% 的请求都是没有意义的。

当使用 WebSocket 推送数据时,上述场景的网络负载为:

5 s 产生一次数据:(2+320)×5×4 = 6 460 bytes = 6.3 kB;

10 s 产生一次数据:(2+320)×5×4 = 39 200 bytes = 6.3 kB;

30 s 产生一次数据:(2+320)×5×4 = 39 200 bytes = 6.3 kB。

观察图 4 可以发现,Ajax 轮询只有在服务器和浏览器间保持相同的刷新频率时,才能保证效率。如果服务器的采集频率很低,而浏览器的轮询频率过高,会产生无效轮询,这些无效轮询会占用大量网络流量,同时会增加服务器的负荷。而使用 WebSocket 时,即使服务器产生的数据的间隔发生了变化,也不需要同步服务器和浏览器间的刷新频率,产生的网络流量也非常平缓。

WebSocket 连接建立完成后,服务器和浏览器采用全双工通信,通信双方能够了解彼此的工作状态。HTTP 协议是面向无状态的(Stateless)短连接协议,浏览器端只能通过发送请求来获取服务器的最新状态,服务器不能直接把数据发送给浏览器端。

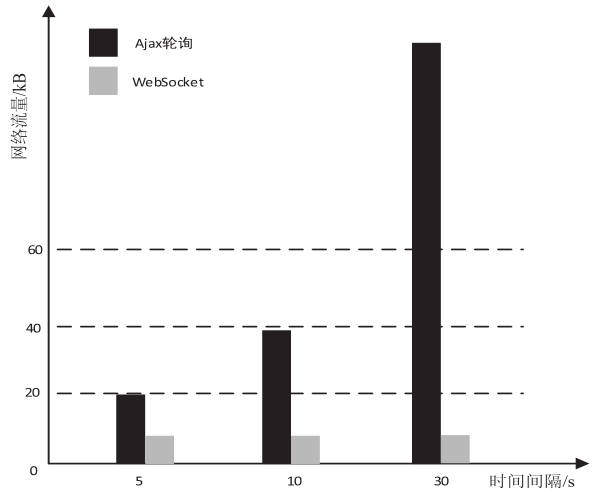

图 4　Ajax 轮询和 WebSocket 网络流量对比

3.2　网络延迟

采用 Ajax 轮询的方法,发送请求的频率是固定的,消息从服务器传输到浏览器端的时间也是固定的,网络延迟时间为 54 ms。实际情况下的网络延迟可能会更大,因为采用轮询的方式,浏览器如果不主动去服务器“拉”数据,服务器不会主动把数据传输给浏览器。如果浏览器“拉”数据的频率过低,浏览器获取数据的时间和采集的时间将会存在一个时间差,这会影响泵站远程监控的决策指挥,但是如果“拉”数据的频率太高,浏览器就会获取到大量的重复数据,这会严重影响服务器的性能,造成服务器和网络资源的浪费。采用 WebSocket 的方法在创建连接的时候,所需要的时间仍然是 54 ms,但是一旦连接建立后,连接会保持打开,服务器会实时将采集到的数据推送到浏览器端,浏览器总是能及时获取到服务器最新的采集数据。

3.3　服务器性能评估

为了验证在 Ajax 轮询模式和 WebSocket 方式下,服务器的压力承载能力,对不同的请求数量进行测试。每个用户可以通过一个线程来模拟,最后对测试数据进行收集、整理和分析。每次只返回一个泵站的实时数据,数据的大小为 16 个字节。

采用单台服务器进行测试。服务器硬件配置:Intel(R) Pentium(R) CPU G2020,内存为 4 GB。Web 服务器采用 tomcat7。

Ajax 轮询服务器性能如表 1 所示。

从表 1 中可以看出,在使用 Ajax 轮询时,服务器每秒处理的请求平均为(5.58+6.54+6.94+6.25+6.54)/5 = 6.37 个。平均内存使用为(139 312+147 920+152 810+150 888+159 360)/5 = 150 058 kB。总运行时间取决于处理请求的线程数量和请求的数量。随着请求数量的增加,总运行时间随之增加,随着处理请求的

线程的数量增加,总的处理时间会减少。

表 1 Ajax 轮询服务器性能表

请求次数	CPU 使用率/%	内存使用/kB	每秒处理请求数量	处理请求线程数量	总运行时间/ms
50	5.1	139 312	5.58	28	8 954
100	7.8	147 920	6.54	32	15 281
200	8.5	152 810	6.95	30	28 790
400	8.4	150 888	6.25	32	63 952
1 000	8.2	159 360	6.54	31	152 816

使用 WebSocket 时服务器的性能如表 2 所示。

表 2 WebSocket 服务器性能表

请求次数	CPU 使用率/%	内存使用/kB	处理请求线程数量	每秒处理请求数量	总运行时间/ms
50	4.9	122 210	29	7.11	7 032
100	7.5	131 821	31	7.19	13 912
200	8.0	137 852	30	7.12	28 091
400	7.8	134 190	32	7.55	52 984
1 000	7.4	140 960	33	8.09	123 601

从表 2 可以看出,在使用 WebSocket 时,服务器每秒处理请求平均为(7.11+7.19+7.12+7.55+8.09)/5=7.412 个。平均内存使用(122 210+131 821+137 852+134 190+140 960)/5=133 406.6 kB。使用 WebSocket 推送数据时,服务器的处理请求的能力有了不少改善,服务器和浏览器之间用于报头的开销明显减少了,服务器处理请求时,用于解析报头的开销也随之减少,所以处理的时间也会减少,每秒处理请求的数量会比 Ajax 轮询要多。在降低服务器内存使用的同时,网络通信的效率也会得到提高。在服务器端请求大量并发的情况下,性能上的优势会有更进一步的体现。

图 5 对比了 Ajax 轮询和 WebSocket 的请求处理能力。从实验结果来看,Ajax 轮询每秒钟处理请求的数量不及 WebSocket。从图中可以看出,在并发请求次数小于 400 时,服务器采用 WebSocket 协议处理请求的数量变化不大,但是采用 Ajax 轮询时,服务器的处理能力会出现抖动。

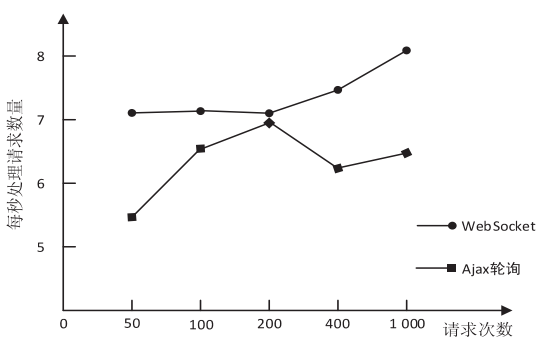

图 5 Ajax 轮询和 WebSocket 处理请求能力对比图

4 结束语

随着互联网技术的发展,Web 实时技术在越来越多场景下发挥重要作用,水利泵站也从以前的人工值守转向自动化控制和远程监控。因此,数据发布的实时性成为了泵站远程监控的重点关注对象。WebSocket 协议允许通过单一 TCP 套接字在 Web 应用程序和 Web 服务器之间实现双向通信。文中在充分对比 Ajax 轮询和 Comet 技术的前提上,将 WebSocket 规范应用到泵站远程监控中,能够有效提高 Web 服务器和浏览器间消息传递的实时性,同时,能够有效降低通信开销,提升服务器性能。

参考文献:

[1] 蒋乾悦,张亚英. 基于模糊综合决策的服务器推送方法[J]. 计算机科学,2014,41(5):86-90.

[2] 李小智. 基于消息中间件的服务器推送技术的应用研究[D]. 长沙:湖南大学,2010.

[3] Bozdag E, Mesbah A, van Deursen A. A comparison of push and pull techniques for Ajax[C]//Proc of 9th IEEE international workshop on web site evolution. [s. l.]:IEEE,2007:15-22.

[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[J]. 计算机系统应用,2008,17(11):116-120.

[5] Li Penghui, Chen Yan, Li Taoying, et al. Research and application of comet technology in public logistics information platform[C]//Proc of international conference on management science and engineering. [s. l.]:[s. n.],2013:152-157.

[6] 陈 航,赵 方. 基于服务器推送技术和 XMPP 的 Web IM 系统实现[J]. 计算机工程与设计,2010,31(5):925-928.

[7] 张 敏,杨海根,贺军义,等. 基于 DWR 的保险项目管理系统的设计与实现[J]. 计算机工程与设计,2011,32(4):1165-1168.

[8] 王建霞,邵永星,张晓明,等. 基于 Pushlet 服务器推送技术的 Web 车辆实时定位系统[J]. 计算机应用与软件,2014,31(4):77-81.

[9] 薛真真. 基于服务器推送和事件流处理技术的实时 Web 系统研究[D]. 杭州:浙江大学,2008.

[10] 林海峰. 基于 Comet 技术的泵站远程监控系统研究与实现[D]. 武汉:武汉理工大学,2010.

[11] 胡晓燕. 基于服务器集群的推送技术的研究与应用[D]. 南京:南京理工大学,2014.

[12] Pimentel V, Nickerson B G. Communicating and displaying real-time data with WebSocket[J]. IEEE Internet Computing, 2012,16(4):45-53.

[13] Chen B, Xu Z. A framework for browser-based multiplayer online games using WebGL and WebSocket[C]//Proc of international conference on multimedia technology. [s. l.]:IEEE, 2011:471-474.

[14] 易仁伟. 基于 WebSocket 的实时 Web 应用的研究[D]. 武汉:武汉理工大学,2013.

面向泵站远程监控的Push技术设计

作者： 周建群， 李千目， ZHOU Jian-qun， LI Qian-mu
作者单位： 南京理工大学 计算机科学与工程学院,江苏 南京,210094
刊名： 计算机技术与发展 ISTIC
英文刊名： Computer Technology and Development
年，卷(期)： 2015(9)