java WebSocket开发入门WebSocket

前言

之前一个项目中九风开发app的用户的消息部分，由于项目比较紧，而且之前没有接触过WebSocket开发，所以暂时先使用轮询方式来开发消息模块，最近准备升级消息模块，准备使用tomcat的WebSocket来开发消息，写此文章方便自己也方便大家。

如需马上测试的scoket的请直接往下翻到代码出。

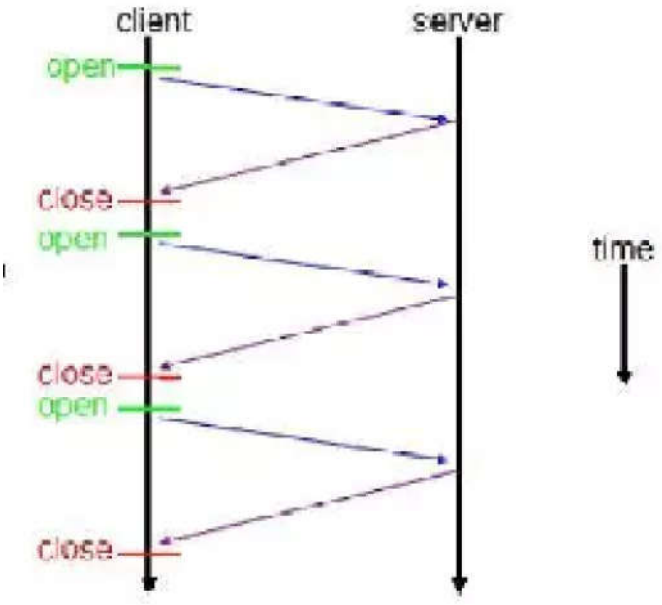
这篇文章中的代码不能运行在spring mvc模式下，如需在mvc模式下运行，请参考这篇Sring MVC 模式下使用websocket (https://www.jianshu.com/p/3398d0230e5f)。

特别说明

此文章中的**后台代码不能直接用于Spring MVC中web层、service层直接调用**，下篇文章准备写这个（还没写好，九风尽快）， 文章中有需要改正的还请简友指出。

消息推送

消息推送大家都不陌生，比如扣扣消息、某东某宝购物后的系统消息等等都是消息推送，在H5出来之前，消息推送基本上都是使用HTTP请求的，但HTTP请求只能在客户端发起请求后服务端返回消息，而不能再客户端未发起请求时服务端主动推送消息给客户端，而对于HTTP的方式实现消息推送时，有以下几种方式：



传统HTTP请求响应客户端服务器的交互图

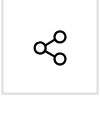
轮询方式：客户端定时向服务端发送ajax请求，服务器接收到请求后马上返回消息并关闭连接。
优点：后端程序编写比较容易。
缺点：TCP的建立和关闭操作浪费时间和带宽，请求中有大半是无用，浪费带宽和服务器资源。
实例：适于小型应用。

长轮询：客户端向服务器发送Ajax请求，服务器接到请求后hold住连接，直到有新消息才返回响应信息并关闭连接，客户端处理完响应信息后再向服务器发送新的请求。
优点：在无消息的情况下不会频繁的请求，耗费资源小。
缺点：服务器hold连接会消耗资源，返回数据顺序无保证，难于管理维护。
实例：WebQQ、Hi网页版、Facebook IM。

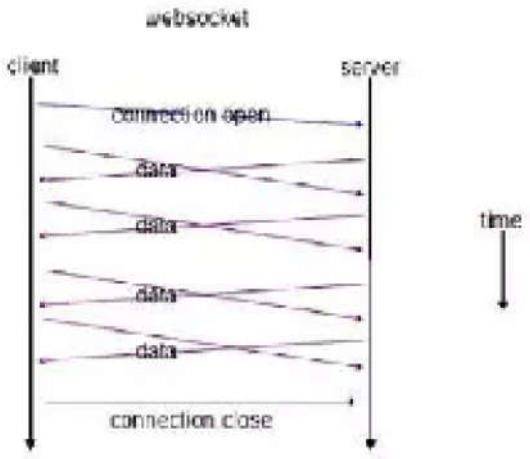
长连接：在页面里嵌入一个隐藏iframe，将这个隐藏iframe的src属性设为对一个长连接的请求或是采用xhr请求，服务器端就能源源不断地往客户端输入数据。
优点：消息即时到达，不发无用请求；管理起来也相对方便。
缺点：服务器维护一个长连接会增加开销，当客户端越来越多的时候，server压力大！
实例：Gmail聊天

Flash Socket：在页面中内嵌入一个使用了Socket类的 Flash 程序JavaScript通过调用此Flash程序提供的Socket接口与服务器端的Socket接口进行通信，JavaScript在收到服务器端传送的信息后控制页面的显示。
优点：实现真正的即时通信，而不是伪即时。
缺点：客户端必须安装Flash插件，移动端支持不好，IOS系统中没有flash的存在；非HTTP协议，无法自动穿越防火墙。
实例：网络互动游戏。

webSocket：HTML5 WebSocket设计出来的目的就是取代轮询和长连接，使客户端浏览器具备像C/S框架下桌面系统的即时通讯能力，实现了浏览器和服务器全双工通信，建立在TCP之上，虽然WebSocket和HTTP一样通过TCP来传输数据，但WebSocket可以主动的向对方发送或接收数据，就像Socket一样；并且WebSocket需要类似TCP的客户端和服务端通过握手连接，连接成功后才能互相通信。
优点：双向通信、事件驱动、异步、使用ws或wss协议的客户端能够真正实现意义上的推送功能。
缺点：少部分浏览器不支持。
示例：社交聊天（微信、QQ）、弹幕、多玩家玩游戏、协同编辑、股票基金实时报价、体育实况更新、视频会议/聊天、基于位置的应用、在线教育、智能家居等高实时性的场景。

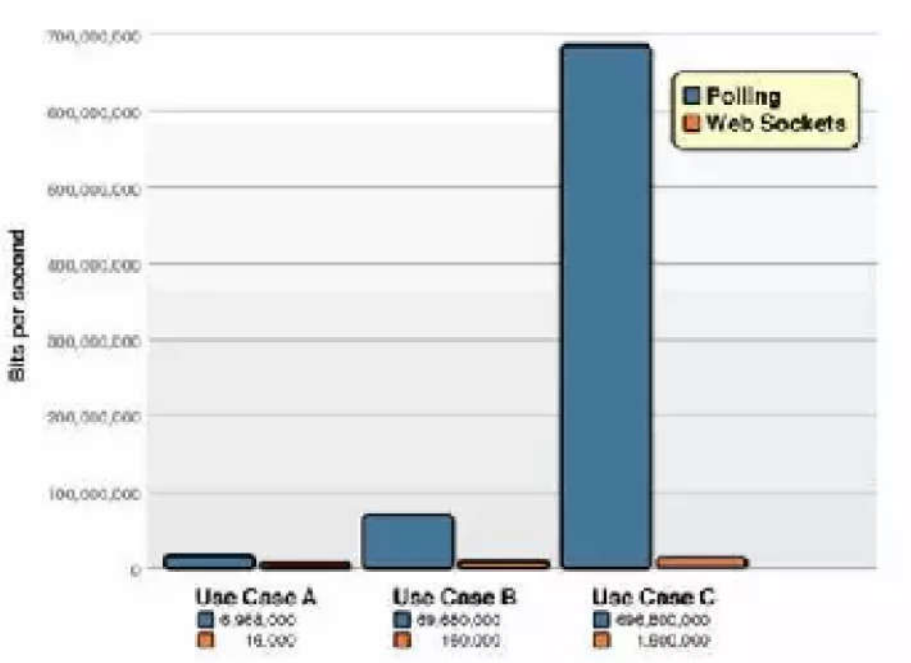


而websocket请求和服务器交互的如下图所示：



WebSocket 请求响应客户端服务器交互图

对比前面的http的客户端服务器的交互图可以发现WebSocket方式减少了很多TCP打开和关闭连接的操作，WebSocket的资源利用率高。



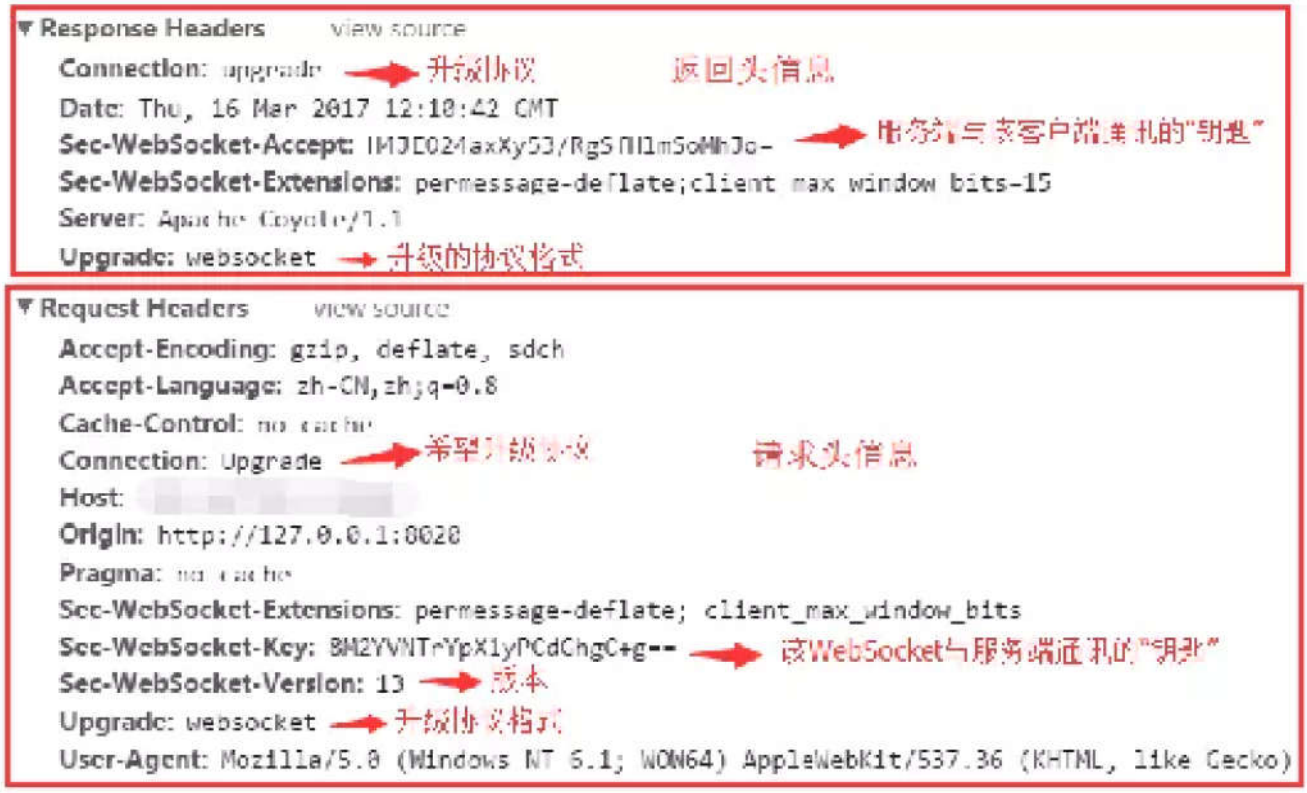
轮询和 WebSocket 实现方式的网络负载对比图

WebSocket规范

WebSocket一种在单个 TCP 连接上进行全双工通讯的协议。WebSocket通信协议于2011年被IETF定为标准RFC 6455，并被RFC7936所补充规范。WebSocket API也被W3C定为标准。

WebSocket 使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在 WebSocket API 中，浏览器和服务器只需要完成一次握手，两者之间就直接可以创建持久性的连接，并进行双向数据传输。

WebSocket 协议本质上是一个基于 TCP 的协议。为了建立一个 WebSocket 连接，客户端浏览器首先要向服务器发起一个 HTTP 请求，这个请求和通常的 HTTP 请求不同，包含了一些附加头信息，附加信息如图所示：



WebSocket请求与响应头内容解析

浏览器支持：所有的最新浏览器支持最新WebSocket规范(RFC 6455 (https://link.jianshu.com?t=https://tools.ietf.org/html/rfc6455))，从维基百科 (https://link.jianshu.com?t=https://zh.wikipedia.org/wiki/WebSocket)上介绍浏览器对WebSocket的支持如下表所示：

浏览器	Chrome	Edge	Firfox	IE	Opera	Safari
最低版本	16	支持	11.0	10	12.10	6.0

移动端支持：移动端基本都支持websocket了，其实和浏览器版支持的版本一样，具体支持如下所示：

最低	android浏览器	Chrome 移动版	Firfox 移动版	Opera 移动版	Safari IOS版

最低	android浏览器	Chrome 移动版	Firfox 移动版	Opera 移动版	Safari IOS版
最低版本	4.4	16	11.0	12.10	6.0

服务器支持：目前主流的web服务器都已经支持，具体版本如下表所示：

厂商	应用服务器	备注
IBM	WebSphere	WebSphere 8.0 以上版本支持，7.X 之前版本结合 MQTT 支持类似的 HTTP 长连接
甲骨文	WebLogic	WebLogic 12c 支持，11g 及 10g 版本通过 HTTP Publish 支持类似的 HTTP 长连接
微软	IIS	IIS 7.0+支持
Apache	Tomcat	Tomcat 7.0.5 + 支持，7.0.2X 及 7.0.3X 通过自定义 API 支持
	Jetty	Jetty 7.0 + 支持

以下内容将使用tomcat服务器来实现Websocket

java WebSocket实现

Oracle 发布的 java 的 WebSocket 的规范是 JSR356规范 (https://link.jianshu.com?t=http://www.oracle.com/technetwork/articles/java/jsr356-1937161.html) ,Tomcat从7.0.27开始支持WebSocket，从7.0.47开始支持JSR-356。

websocket简单实现分为以下几个步骤：**添加websocket库**、**编写后台代码**、**编写前端代码**。

添加websocket库

在maven中添加websocket库的代码如下所示：

```
<dependency>
   <groupId>javax.websocket</groupId>
   <artifactId>javax.websocket-api</artifactId>
   <version>1.1</version>
   <scope>provided</scope>
</dependency>
```

九风有次没写<scope>字段，前端后台都会报错，大家记得加上就行。
前端错误内容：
WebSocket connection to 'ws://localhost:8080/ (https://link.jianshu.com?

t=ws://localhost:8080/){project-name}/websocket' failed: Error during WebSocket handshake: Unexpected response code: 404" 。

后台错误内容：

Did not find handler method for [/websocket]
Matching patterns for request [/websocket] are [/**]
URI Template variables for request [/websocket] are {}
Mapping [/websocket] to HandlerExecutionChain with handler [org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler@398f0b1f] and 1 interceptor
Last-Modified value for [/{project-name}/websocket] is: -1

编写后台代码

后台实现websocket有两种方式：使用继承类、使用注解；注解方式比较方便，一下代码中使用注解方式来进行演示。

声明websocket地址类似Spring MVC中的@controller注解类似，websocket使用@ServerEndpoint来进行声明接口： `@ServerEndpoint(value="/websocket/{paraName}")` ；其中 “ { } ”用来表示带参数的连接，如果需要获取{}中的参数在参数列表中增加：@PathParam("paraName") Integer userId 。则连接地址形如：ws://localhost:8080/project-name/websocket/8 (https://link.jianshu.com?t=ws://localhost:8080/project-name/websocket/8)，其中每个连接可以设置不同的paraName的值。

注解、成员数据介绍：

1.@OnOpen
public void onOpen(Session session) throws IOException{ } -------有连接时的触发函数。 我们可以在用户连接时记录用户的连接带的参数，只需在参数列表中增加参数：@PathParam("paraName") String paraName。

2.@OnClose
public void onClose(){ } ------连接关闭时的调用方法。

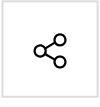
3.@OnMessage
public void onMessage(String message, Session session) { } -------收到消息时调用的函数，其中Session是每个websocket特有的数据成员，详情见4.

4.Session ----每个Session代表了两个web socket断点的会话；当websocket握手成功后，websocket就会提供一个打开的Session，可以通过这个Session来对另一个端点发送数据；如果Session关闭后发送数据将会报错。

5.Session.getBasicRemote().sendText("message") ------向该Session连接的用户发送字符串数据。

6.@OnError
public void onError(Session session, Throwable error) { } -------发生意外错误时调用的函数。

后台代码：有以上基础后就直接上代码了.

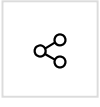


```
import java.io.IOException;
import javax.websocket.OnClose;
import javax.websocket.OnError;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.Session;
import javax.websocket.server.PathParam;
import javax.websocket.server.ServerEndpoint;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
/**
 * @Class: Test
 * @Description: 简单websocket demo
 * @author 九风萍舟
 */
@ServerEndpoint(value="/websocketTest/{userId}")
public class Test {
    private Logger logger = LoggerFactory.getLogger(Test.class);

    private static String userId;

    //连接时执行
    @OnOpen
    public void onOpen(@PathParam("userId") String userId,Session session) throws IOE
        this.userId = userId;
        logger.debug("新连接：{}",userId);
    }

    //关闭时执行
    @OnClose
    public void onClose(){
        logger.debug("连接：{} 关闭",this.userId);
    }

    //收到消息时执行
    @OnMessage
    public void onMessage(String message, Session session) throws IOException {
        logger.debug("收到用户{}的消息{}",this.userId,message);
        session.getBasicRemote().sendText("收到 "+this.userId+" 的消息 "); //回复用户
    }

    //连接错误时执行
    @OnError
    public void onError(Session session, Throwable error){
        logger.debug("用户id为：{}的连接发送错误",this.userId);
        error.printStackTrace();
    }

}
```

ServerEndpoint报错： 原因是不能自动检测 ServerEndpoint 的包，解决方法：复制 `import javax.websocket.server.ServerEndpoint;` 到文件程序 import 区域即可。

编写前端代码

后台代码编写了那么前端代码就几乎不用讲解了，相信大家一眼就能看得懂。

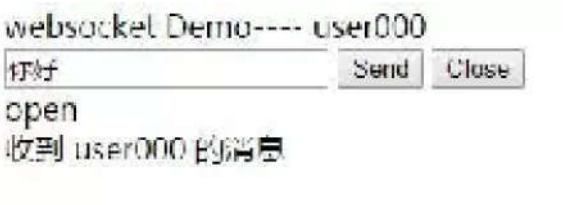


```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        websocket Demo---- user000 <br />
        <input id="text" type="text" />
        <button onclick="send()"> Send </button>
        <button   onclick="closeWebSocket()"> Close </button>
        <div id="message">   </div>

    <script type="text/javascript">
     //判断当前浏览器是否支持WebSocket
      if('WebSocket' in window){
          websocket = new WebSocket("ws://localhost:8080/Demo/websocketTest/user000")
          console.log("link success")
      }else{
          alert('Not support websocket')
      }

      //连接发生错误的回调方法
      websocket.onerror = function(){
          setMessageInnerHTML("error");
      };

      //连接成功建立的回调方法
      websocket.onopen = function(event){
          setMessageInnerHTML("open");
      }
       console.log("-----")
      //接收到消息的回调方法
      websocket.onmessage = function(event){
            setMessageInnerHTML(event.data);
      }

      //连接关闭的回调方法
      websocket.onclose = function(){
          setMessageInnerHTML("close");
      }

      //监听窗口关闭事件，当窗口关闭时，主动去关闭websocket连接，防止连接还没断开就关闭窗口
      window.onbeforeunload = function(){
          websocket.close();
      }

      //将消息显示在网页上
      function setMessageInnerHTML(innerHTML){
          document.getElementById('message').innerHTML += innerHTML + '<br/>';
      }

      //关闭连接
      function closeWebSocket(){
          websocket.close();
      }

      //发送消息
```

```
        function send(){
            var message = document.getElementById('text').value;
            websocket.send(message);
        }
    </script>

    </body>
</html>
```

测试运行

在Chrome上打开前端代码后，马上就建立了连接，大家可以使用F12查看下建立连接的请求与响应，可以对比前面关于协议建立的部分进行学习。
建立连接后，想后台发送数据后，同时可以看到后台返回的信息：



前端测试

在后台可以看到连接的建立和收到的数据：

后台测试

对于其他功能功能大家可以自己测测。

总结

websocket特别适合于需要实时数据传送的场景，比轮询方式效率高很多。

参考

(https://link.jianshu.com?t=https://www.ibm.com/developerworks/cn/java/j-lo-WebSocket/)WebSocket与消息推送 (https://link.jianshu.com?t=http://www.cnblogs.com/hz1124/p/6094890.html)
(https://link.jianshu.com?t=https://tools.ietf.org/html/rfc6455)Java后端WebSocket的Tomcat实现 (https://link.jianshu.com?t=http://www.cnblogs.com/xdp-gacl/p/5193279.html?utm_source=tuicool&utm_medium=referral)
WebSocket 实战 (https://link.jianshu.com?

t=https://www.ibm.com/developerworks/cn/java/j-lo-WebSocket/)
使用 HTML5 WebSocket 构建实时 Web 应用 (https://link.jianshu.com?
t=https://www.ibm.com/developerworks/cn/web/1112_huangxa_websocket/)
混合移动应用的消息推送之 websocket (https://link.jianshu.com?
t=https://www.ibm.com/developerworks/cn/mobile/mo-cn-websocket/index.html)
WebSocket 维基百科 (https://link.jianshu.com?
t=https://zh.wikipedia.org/wiki/WebSocket)
WebSocket 接口文档 (https://link.jianshu.com?
t=http://docs.oracle.com/javaee/7/api/javax/websocket/package-summary.html)
RFC 6455 规范 (https://link.jianshu.com?t=https://tools.ietf.org/html/rfc6455)
JSR 356, Java API for WebSocket (https://link.jianshu.com?
t=http://www.oracle.com/technetwork/articles/java/jsr356-1937161.html)