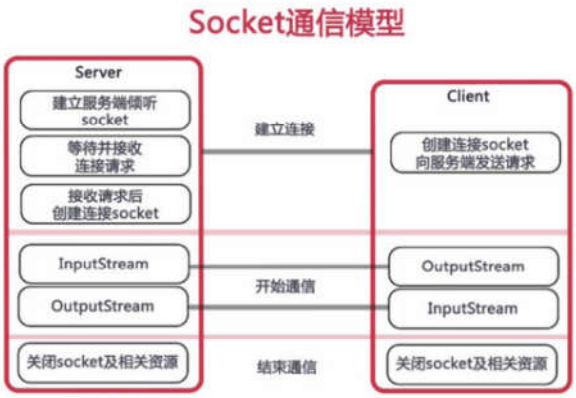
WebSocket与消息推送

目录

B/S结构的软件项目中有时客户端需要实时的获得服务器消息，但默认HTTP协议只支持请求响应模式，这样做可以简化Web服务器，减少服务器的负担，加快响应速度，因为服务器不需要与客户端长时间建立一个通信链接，但不容易直接完成实时的消息推送功能，如聊天室、后台信息提示、实时更新数据等功能，但通过polling、Long polling、长连接、Flash Socket以及HTML5中定义的WebSocket能完成该功能需要。

一、Socket简介

Socket又称"套接字"，应用程序通常通过"套接字"向网络发出请求或者应答网络请求。Socket的英文原义是“孔”或“插座”，作为UNIX的进程通信机制。Socket可以实现应用程序间网络通信。



Socket可以使用TCP/IP协议或UDP协议。

TCP/IP协议

TCP/IP协议是目前应用最为广泛的协议，是构成Internet国际互联网协议的最为基础的协议,由TCP和IP协议组成:
TCP协议:面向连接的、可靠的、基于字节流的传输层通信协议，负责数据的可靠性传输的问题。

IP协议:用于报文交换网络的一种面向数据的协议，主要负责给每台网络设备一个网络地址，保证数据传输到正确的目的地。

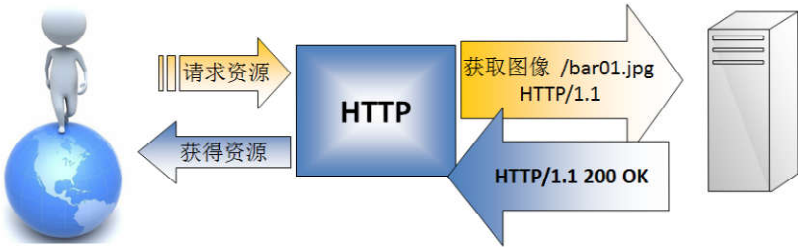
UDP协议

UDP特点：无连接、不可靠、基于报文的传输层协议，优点是发送后不用管，速度比TCP快。

二、WebSocket简介与消息推送

B/S架构的系统多使用HTTP协议，HTTP协议的特点：

1 无状态协议
2 用于通过 Internet 发送请求消息和响应消息
3 使用端口接收和发送消息，默认为80端口
底层通信还是使用Socket完成。



HTTP协议决定了服务器与客户端之间的连接方式，无法直接实现消息推送（F5已坏）,一些变相的解决办法：

双向通信与消息推送

轮询：客户端定时向服务器发送Ajax请求，服务器接到请求后马上返回响应信息并关闭连接。 优点：后端程序编写比较容易。 缺点：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。 Comet异步的ashx， 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；管理起来也相对便。 缺点：服务器维护一个长连接会增加开销。 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；非HTTP协议，无法自动穿越防火墙。 实例：网络互动游戏。

Websocket:
WebSocket是HTML5开始提供的一种浏览器与服务器间进行全双工通讯的网络技术。依靠这种技术可以实现客户端和服务器端的长连接，双向实时通信。
特点:
事件驱动

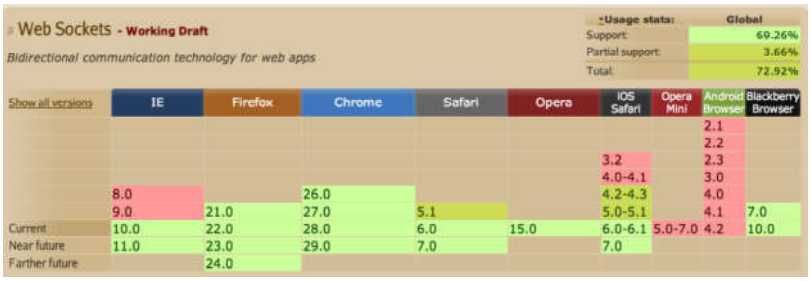
异步

使用ws或者wss协议的客户端socket

能够实现真正意义上的推送功能

缺点：

少部分浏览器不支持，浏览器支持的程度与方式有区别。



三、WebSocket客户端

websocket允许通过JavaScript建立与远程服务器的连接，从而实现客户端与服务器间双向的通信。在websocket中有两个方法：

1、send() 向远程服务器发送数据

2、close() 关闭该websocket链接

websocket同时还定义了几个监听函数

1、onopen 当网络连接建立时触发该事件

2、onerror 当网络发生错误时触发该事件

3、onclose 当websocket被关闭时触发该事件

4、onmessage 当websocket接收到服务器发来的消息的时触发的事件，也是通信中最重要的一个监听事件。msg.data

websocket还定义了一个readyState属性，这个属性可以返回websocket所处的状态：

1、CONNECTING(0) websocket正尝试与服务器建立连接

2、OPEN(1) websocket与服务器已经建立连接

3、CLOSING(2) websocket正在关闭与服务器的连接

4、CLOSED(3) websocket已经关闭了与服务器的连接

websocket的url开头是ws，如果需要ssl加密可以使用wss，当我们调用websocket的构造方法构建一个websocket对象（new WebSocket(url)）的之后，就可以进行即时通信了。

```
<!DOCTYPE html>
<html>

  <head>
    <meta name="viewport" content="width=device-width" />
    <title>WebSocket 客户端</title>
  </head>

  <body>
    <div>
      <input type="button" id="btnConnection" value="连接" />
      <input type="button" id="btnClose" value="关闭" />
```

```
        <input type="button" id="btnSend" value="发送" />
    </div>
    <script src="js/jquery-1.11.1.min.js" type="text/javascript" charset="utf-8"></script
>
    <script type="text/javascript">
        var socket;
        if(typeof(WebSocket) == "undefined") {
            alert("您的浏览器不支持WebSocket");
            return;
        }

        $("#btnConnection").click(function() {
            //实现化WebSocket对象，指定要连接的服务器地址与端口
            socket = new WebSocket("ws://192.168.1.2:8888");
            //打开事件
            socket.onopen = function() {
                alert("Socket 已打开");
                //socket.send("这是来自客户端的消息" + location.href + new Date());
            };
            //获得消息事件
            socket.onmessage = function(msg) {
                alert(msg.data);
            };
            //关闭事件
            socket.onclose = function() {
                alert("Socket已关闭");
            };
            //发生了错误事件
            socket.onerror = function() {
                alert("发生了错误");
            }
        });

        //发送消息
        $("#btnSend").click(function() {
            socket.send("这是来自客户端的消息" + location.href + new Date());
        });

        //关闭
        $("#btnClose").click(function() {
            socket.close();
        });
    </script>
  </body>

</html>
```

四、WebSocket服务器端

JSR356定义了WebSocket的规范，Tomcat7中实现了该标准。JSR356 的 WebSocket 规范使用 javax.websocket.*的 API，可以将一个普通 Java 对象（POJO）使用 @ServerEndpoint 注释作为 WebSocket 服务器的端点。

```
@ServerEndpoint("/push")
 public class EchoEndpoint {

 @OnOpen
 public void onOpen(Session session) throws IOException {
 //以下代码省略...
 }

 @OnMessage
 public String onMessage(String message) {
 //以下代码省略...
 }

 @Message(maxMessageSize=6)
 public void receiveMessage(String s) {
 //以下代码省略...
 }

 @OnError
 public void onError(Throwable t) {
 //以下代码省略...
 }

 @OnClose
 public void onClose(Session session, CloseReason reason) {
 //以下代码省略...
 }

 }
```

上面简洁代码即建立了一个WebSocket的服务端，@ServerEndpoint("/push")的annotation注释端点表示将WebSocket服务端运行在ws://[Server端IP或域名]:[Server端口]/项目/push的访问端点，客户端浏览器已经可以对WebSocket客户端API发起HTTP长连接了。

使用ServerEndpoint注释的类必须有一个公共的无参数构造函数，@onMessage注解的Java方法用于接收传入的WebSocket信息，这个信息可以是文本格式，也可以是二进制格式。

OnOpen在这个端点一个新的连接建立时被调用。参数提供了连接的另一端的更多细节。Session表明两个WebSocket端点对话连接的另一端，可以理解为类似HTTPSession的概念。

OnClose在连接被终止时调用。参数closeReason可封装更多细节，如为什么一个WebSocket连接关闭。

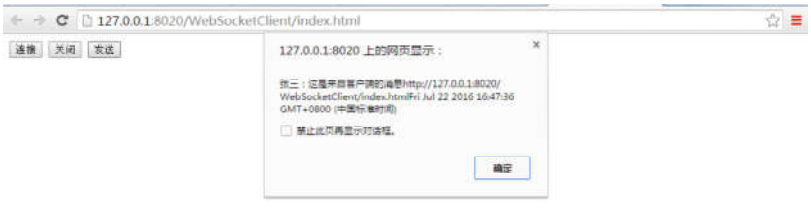
更高级的定制如@Message注释，MaxMessageSize属性可以被用来定义消息字节最大限制，在示例程序中，如果超过6个字节的信息被接收，就报告错误和连接关闭。

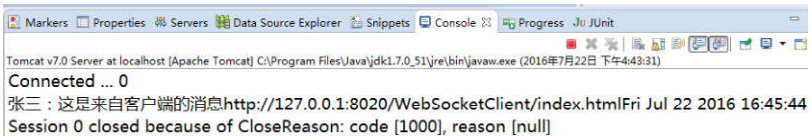
```
package action;

import javax.websocket.CloseReason;
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;

//ws://127.0.0.1:8087/Demo1/ws/张三
@ServerEndpoint("/ws/{user}")
public class WSServer {
    private String currentUser;

    //连接打开时执行
    @OnOpen
    public void onOpen(@PathParam("user") String user, Session session) {
        currentUser = user;
        System.out.println("Connected ... " + session.getId());
    }

    //收到消息时执行
    @OnMessage
    public String onMessage(String message, Session session) {
        System.out.println(currentUser + " : " + message);
        return currentUser + " : " + message;
    }

    //连接关闭时执行
    @OnClose
    public void onClose(Session session, CloseReason closeReason) {
        System.out.println(String.format("Session %s closed because of %s", session.getId(),
closeReason));
    }

    //连接错误时执行
    @OnError
    public void onError(Throwable t) {
        t.printStackTrace();
    }
}
```

url中的字符张三是的路径参数，响应请求的方法将自动映射。

五、测试运行





六、小结与消息推送框架

Socket在应用程序间通信被广泛使用，如果需要兼容低版本的浏览器，建议使用反向ajax或长链接实现；如果纯移动端或不需考虑非现代浏览器则可以直接使用websocket。Flash实现推送消息的方法不建议使用，因为依赖插件且手机端支持不好。关于反向ajax也有一些封装好的插件如“Pushlet”

6.1、开源Java消息推送框架 Pushlet

Pushlet 是一个开源的 Comet 框架,Pushlet 使用了观察者模型：客户端发送请求，订阅感兴趣的事件；服务器端为每个客户端分配一个会话 ID 作为标记，事件源会把新产生的事件以多播的方式发送到订阅者的事件队列里。

源码地址：https://github.com/wjw465150/Pushlet

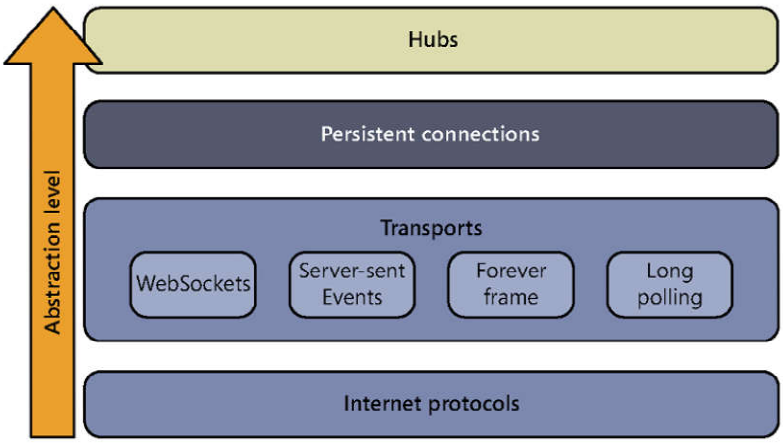
Pushlet是一种comet实现：在Servlet机制下，数据从server端的Java对象直接推送（push）到（动态）HTML页面，而无需任何Javaapplet或者插件的帮助。它使server端可以周期性地更新client的web页面，这与传统的request/response方式相悖。浏览器client为兼容JavaScript1.4版本以上的浏览器（如InternetExplorer、FireFox），并使用JavaScript/DynamicHTML特性。而底层实现使用一个servlet通过Http连接到JavaScript所在的浏览器，并将数据推送到后者。

6.2、开源DotNet消息推送框架SignalR

SignalR是一个ASP .NET下的类库，可以在ASP .NET的Web项目中实现实时通信。在Web网页与服务器端间建立Socket连接，当WebSockets可用时（即浏览器支持Html5）SignalR使用WebSockets，当不支持时SignalR将使用长轮询来保证达到相同效果。

官网：http://signalr.net/

源码：https://github.com/SignalR/SignalR



七、代码下载

7.1、Java实现的服务器端代码与客户端代码下载

点击下载服务器端代码

点击下载客户端代码

7.2、DotNet服务器端手动连接实现代码下载

点击下载DotNet服务器端手动连接实现代码

7.3、DotNet下使用SuperWebSocket三方库实现代码下载

点击下载DotNet下使用SuperWebSocket三方库实现代码