链滴

springboot 整合 websocket 技术

作者：FILWLL

原文链接：https://ld246.com/article/1548488644195

来源网站：链滴

什么是Webscoket

简单来说就是一个基于TCP的持久化的网络通信协议。主要作用就是：服务端可以主动推送信息给客端，不需要客户端重复的向服务端发请求查询。

具体详细说明：https://www.zhihu.com/question/20215561 （PS：写的是真的简单易懂）

springboot整合Webscoket

实现websocket有多种方式：最简单的H5， sockjs以及使用STOMP协议

开发环境

1. JDK版本1.8
2. springboot版本2.1.0
3. 开发工具：IDEA MVAEN

基于H5的websocket实现

第一步 引入依赖

在springboot2.0版本之后，官方添加了Webscoket的依赖，因此只需要在pom当中引入

```
 <!-- 引入websocket-->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-websocket</artifactId>
```

```
        </dependency>
```

第二步 配置WebSocket

```
/**
 * @Auther: HJLJY
 * @Date: 2019/1/25 0025 16:45
 * @Description:  websocket客户端配置类
 *
 * @EnableWebSocket   表示该类支持websocket
 * @EnableWebSocketMessageBroker   websocket使用STOMP协议进行消息的传递  还需要进行外的配置不然会报错
 */
@Configuration
@EnableWebSocket
public class WebSocketConfig {
    @Bean
    public ServerEndpointExporter serverEndpointExporter() {
        return new ServerEndpointExporter();
    }
}
```

第三步 设置服务端地址

```
/**
 * @Auther: HJLJY
 * @Date: 2019/1/25 0025 16:56
 * @Description:
 * @ServerEndpoint 用于配置websocket的地址    ws：//127.0.0.1:8080/websocket
 */
@ServerEndpoint(value = "/websocket")
@Component
public class WebSocketServer {
    //TODO 这里根据实际业务进行代码的书写

    //concurrent包的线程安全Set，每个客户端的websocket连接会创建一个WebSocketServer对象

    private static CopyOnWriteArraySet<WebSocketServer> webSocketSet = new CopyOnWrit
ArraySet<>();

    //这个session是每个websocket与某个客户端的连接会话，通过它与客户端进行数据交互。
    private Session session;

    /**
     * websocket连接建立成功调用的方法
     */
    @OnOpen
    public void onOpen(Session session) {
        this.session = session;
        webSocketSet.add(this);     //加入set中
        System.out.println("有新连接加入！ sessionID为：" + session.getId());
        try {
            sendMessage("恭喜 连接websocket服务端成功");
```

```
        } catch (IOException e) {
            System.out.println("IO异常");
        }
    }

    /**
     * 关闭websocket连接调用的方法
     */
    @OnClose
    public void onClose() {
        webSocketSet.remove(this);  //从set中删除
        System.out.println("有一连接关闭！sessionID为：" + session.getId());
    }

    /**
     * 收到客户端消息后调用的方法
     *
     * @param message 客户端发送过来的消息
     */
    @OnMessage
    public void onMessage(String message) {
        System.out.println("来自客户端的消息:" + message);
        //将接受到的消息群发给所有websocket连接。这里也可以根据实际需求修改代码，将接受到的
息发送给某个指定的websocket连接。
        for (WebSocketServer item : webSocketSet) {
            try {
                item.sendMessage(message);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * 发生错误时调用
     */
    @OnError
    public void onError(Session session, Throwable error) {
        System.out.println("发生错误");
        error.printStackTrace();
    }

    public void sendMessage(String message) throws IOException {
        this.session.getBasicRemote().sendText(message);  //这两个方法是同步和异步的关系
        //this.session.getAsyncRemote().sendText(message);
    }
}
```

第四步 客户端实现

客户端的常见api

onopen ：用于建立连接成功后的回调

onclose : 用于关闭连接后的回调

onmessage： 用于接受到消息后的回调

send()： 用于发送消息

具体详细说明：http://www.ruanyifeng.com/blog/2017/05/websocket.html

HTML具体代码：

```
<!DOCTYPE HTML>
<html>
<head>
    <title>测试websocket</title>
</head>

<body>
TEST websocket<br/>
<input id="text" type="text"/>
<button onclick="send()">Send</button>

<button onclick="closeWebSocket()">Close</button>
<div id="message">
</div>
</body>

<script type="text/javascript">
    //这一步就是向后台建立连接    协议开头不是HTTP 而是ws或者wss
    var websocket = new WebSocket("ws://localhost:8080/websocket");

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

    //发送消息
    function send() {
        var message = document.getElementById('text').value;
        websocket.send(message);
    }

    //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("开启连接");
    }

    //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("关闭连接");
    }

    //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server端会抛异常。
    window.onbeforeunload = function () {
        websocket.close();
    }

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


</script>
</html>
```

最后启动springboot项目进入到刚才的html界面（开启两个浏览器同时进入进行测试）查看效果就以了。

参考资料

简单实现：https://www.cnblogs.com/bianzy/p/5822426.html

使用spring boot +WebSocket实现（后台主动）消息推送：https://blog.csdn.net/zhangdehua67/article/details/78913839/

基于STOMP协议：https://www.jianshu.com/p/19cec6fbf422