WebSocket分析、运用与实例

如今，各行各业对于信息的实时性要求越来越高，也出现了许多实用性技术，这里我们就讨论Websocket能做的事。

> 在了解websocket之前，我们先看下通讯方面的几个知识：
>
> - **全双工通信**：又称为双向同时通信，即通信的双方可以同时发送和接收信息的信息交互方式。
> - **半双工通信**：按字面意思也可理解了。
> - **http协议**：超文本传输协议（HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议。所有的WWW文件都必须遵守这个标准。设计HTTP最初的目的是为了提供一种发布和接收HTML页面的方法。1960年美国人Ted Nelson构思了一种通过计算机处理文本信息的方法，并称之为超文本（hypertext）,这成为了HTTP超文本传输协议标准架构的发展根基。
> - **长连接**：在一个连接上可以连续发送多个数据包，在连接保持期间，如果没有数据包发送，需要双方发链路检测包。
> - **短连接**：通讯双方有数据交互时，就建立一个连接，数据发送完成后，则断开此连接，即每次连接只完成一项业务的发送。
> - **轮询（Polling）**：是一种CPU决策如何提供周边设备服务的方式，又称“程控输出入”（Programmed I/O）。轮询法的概念是，由CPU定时发出询问，依序询问每一个周边设备是否需要其服务，有即给予服务，服务结束后再问下一个周边，接着不断周而复始。
> - 无状态协议：无状态协议是指比如客户获得一张网页之后关闭浏览器，然后再一次启动浏览器，再登陆该网站，但是服务器并不知道客户关闭了一次浏览器。

类型	轮询（长/短轮询不做详解）	websocket
浏览器支持	几乎所有浏览器	IE 10+， Edge， Firefox 4+， Chrome 4+， Safari 5+， Opera 11.5+，可自行判断（window.WebSocket，boolean值）
服务器负载	占用较多带宽与内存	占用资源视客户的事件情况决定，性能佳
客服端负载	占用较多内存，发送较多请求数	毫无压力
延迟	有一定延迟，以轮询间隔为准	实时收发信息
实现复杂度	非常简单	客户端与服务端都需设计后编写

websocket原理

首先我们来看下Websocket握手。

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

首先，Sec-WebSocket-Key 是一个Base64 encode的值，这个是浏览器随机生成的，验证是否为websocket服务，第二个参数为一个用户定义的字符串，用来区分同URL下，不同的服务所需要的协议，第三个参数为协议版本。

服务端返回以下内容时，表示连接成功了：

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

简单介绍之后，我们来看看，我们可以使用websocket干些啥呢。

websocket作用

通过对websocket的了解，我们可以发散自己的思维，用websocket做些什么呢？

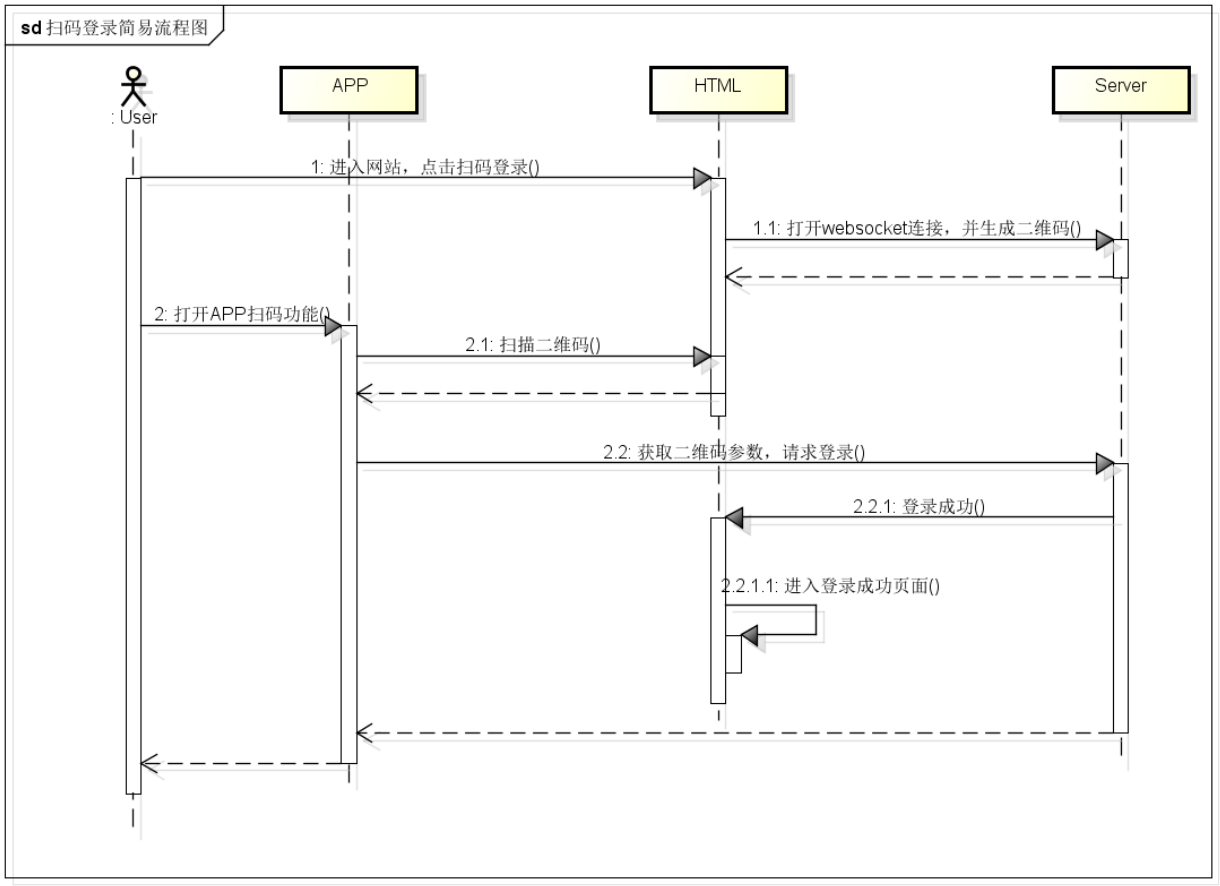
websocket最主要的就是实时性，并且互动性，围绕这2个思路发散下思维。

A与B连接后，可以相互通讯，可以实时监控对方状态，可以相互调用对方的服务，可以即时的查看内容，等等，那么你是否有很多想法要去实现呢？

相信大家都使用过扫码登录，下面我们先来看一个websocket的应用场景，扫码登录功能。

websocket扫码登录实例

真实技术场景：



在简易的流程内，可以看到多方面的交互：

- H端与server端
- APP端与H端
- APP端与server端
- server端与H端

看下H端代码（可以判断浏览器是否支持websocket）：

```
// 连接webSocket服务端，安全连接时使用：
```

```
        onError(evt) // 发生错误事件，可执行需要的逻辑
    };
```

这里的几个方法，对应各自事件。此处以登录扫码为例。

- 首先，在打开时，可发送服务器端，此连接需要登录使用，获取相应参数：

```
websocket.send("login"); // 标志登录用二维码
```

- 接收消息的处理，可看注释理解，登录模块共有3个判断逻辑，**二维码生成、显示图片、登录成功**：

  data.type：请注意此参数，与server交互使用，很重要。需要约定清楚。

```
  if(data.type == "urlParam"){// 前后端约定标志字符串，此处标志为需要生成二维码
        // 获取信息，生成二维码
        var qrcode = new QRCode(document.getElementById("geneimg"), {
            width : 200,//设置宽高
            height : 200
        });
        qrcode.makeCode(JSON.stringify(data.data));

document.getElementById("geneimgdiv").style.display="block"; // 显示二维码

document.getElementById("loginform").style.display="none";//隐藏账号登录内容
  }else if(data.type == "pic"){// 前后端设定标志字符串，此处标志为需要显示用户头像
        var html = "<img width=200 height=200 src=\""+data.msg+"\"><br/>请确认登录";
        document.getElementById("geneimg").innerHTML=html;//
```

```
            if(websocket != null){
                websocket.close();
            }
        }
```

H端拥有以上逻辑。

看下APP端的业务逻辑：

请求登录：

```
// ret为扫码返回的数据，通过返回数据获取二维码内部的参数（扫码功能需要移动端支持，可自行学习，无条件的可用请求模拟：发送请求，附带参数；推荐jmeter或网页内等工具）
var content = JSON.parse(ret.content);
var data = JSON.parse(content);
var type = data.type;
// 二维码中包含类型为"login"，登录参数，并做登录逻辑处理
if(type && type == "login"){
    var uuid = data.uuid;
    var token = $api.getStorage("token");
    // 发送请求，携带APP内登录的token，并将参数一起发送请求
    // 此处将ajax请求统一封装为携带token和不携带token的2种
    // loginsuccess为调用ajax方法后的回调方法
    posttoken("http://localhost:8080/socket/login",
{token:token,type:"login",uuid:uuid},loginsuccess);
    return;
}

// 代码片段 ------start   封装ajax
片段1，在ajax内，加上消息头
headers : {
    "Content-Type" : "application/json;charset=UTF-8",
    // 在请求头信息内放入token，spring security内会填充用户信息，此
```

```
// 携带token，以及uuid（查询webSocket的session信息，以通知浏览器）
posttoken("http://localhost:8080/socket/loginsure",
{token:token,uuid:uuid},loginsuccess);

// 确认登录成功后，关闭APP内页面，APP端结束
```

看下Java服务端的业务逻辑：

提供一个连接类，**此地址不是以映射的形式提供给外部的，而是以资源的形式提供访问**

```
@ServerEndpoint(value = "/websocket")
public class WebSocketController {
    @OnOpen
    public void onOpen(Session session) {}
    @OnMessage
    public void onMessage(Session session, String message) {}
    @OnClose
    public void onClose(Session session) {}
    @OnError
    public void onError(Session session, Throwable error) {}
}
```


以上为服务端的方法，在讲解需要的逻辑之前，首先看下一个工具类，主要管理webSocket的session保存以及获取。

```
public class WebSocketUtils {
    /**
     * 存放session的集合
     * 对应uuid
     */
    private static Map<Session, String> map = new
```

```
            return;
        }
        String uuid = UUIDUtil.getLoUUID();
        map.put(session, uuid);
        revertMap.put(uuid, session);
    }
    /**
     * 链接退出
     * @param session
     */
    public static void offlinePerson(Session session) {
        revertMap.remove(MapUtils.getString(map, session));
        map.remove(session);
    }
    /**
     * 通过uuid查询session
     */
    public static Session getSession(String key) {
        return revertMap.get(key);
    }
    /**
     * 通过session获取uuid
     */
    public static String getUUid(Session session) {
        return MapUtils.getString(map, session);
    }
    /**
     * 向session发送信息
     * @param message
     * @throws IOException
     */
    public static void sendMessage(Session session,
WebsocketModel message) throws IOException {

session.getBasicRemote().sendText(JSONObject.toJSONString(message
));
    }
    /**
```

挨个看下每个方法的逻辑：

```
    @OnOpen
    public void onOpen(Session session) {
        // 每个新连接都存储
        WebSocketUtils.onlinePerson(session);
    }
    @OnMessage
    public void onMessage(Session session, String message) {
        try {
            // 判断是否为登录标识（枚举存放变量）
            if (WebSocketTypeEnum.LOGIN.typeCode.equals(message))
{
                // 获取登录需要的参数
                String urlParam = getUrl(message, session);
                WebsocketModel websocketModel = new
WebsocketModel();

websocketModel.setMsg(WebSocketUtils.getUUid(session));
                websocketModel.setType("urlParam");
                websocketModel.setData(urlParam);
                WebSocketUtils.sendMessage(session,
websocketModel);
            }// 其他类型可以自行尝试：如聊天频道，或添加好友等
        } catch (Exception e) {
            logger.error("发送信息失败，链接关闭" + e.getMessage(),
e);
        }
    }

    private String getUrl(String type, Session session) {
        String uuid = WebSocketUtils.getUUid(session);
        String url = "";
        switch (type) {
            case "login":
                // 类型为登录，并需要通过uuid来确定是哪个浏览器的
session（app扫码后请求需要传递）
```

```
            } catch (Exception e) {
                logger.error("连接关闭" + e.getMessage(), e);
            }
        }
```

以上为H端与java端的交互逻辑，下面看下APP端请求的server端逻辑；

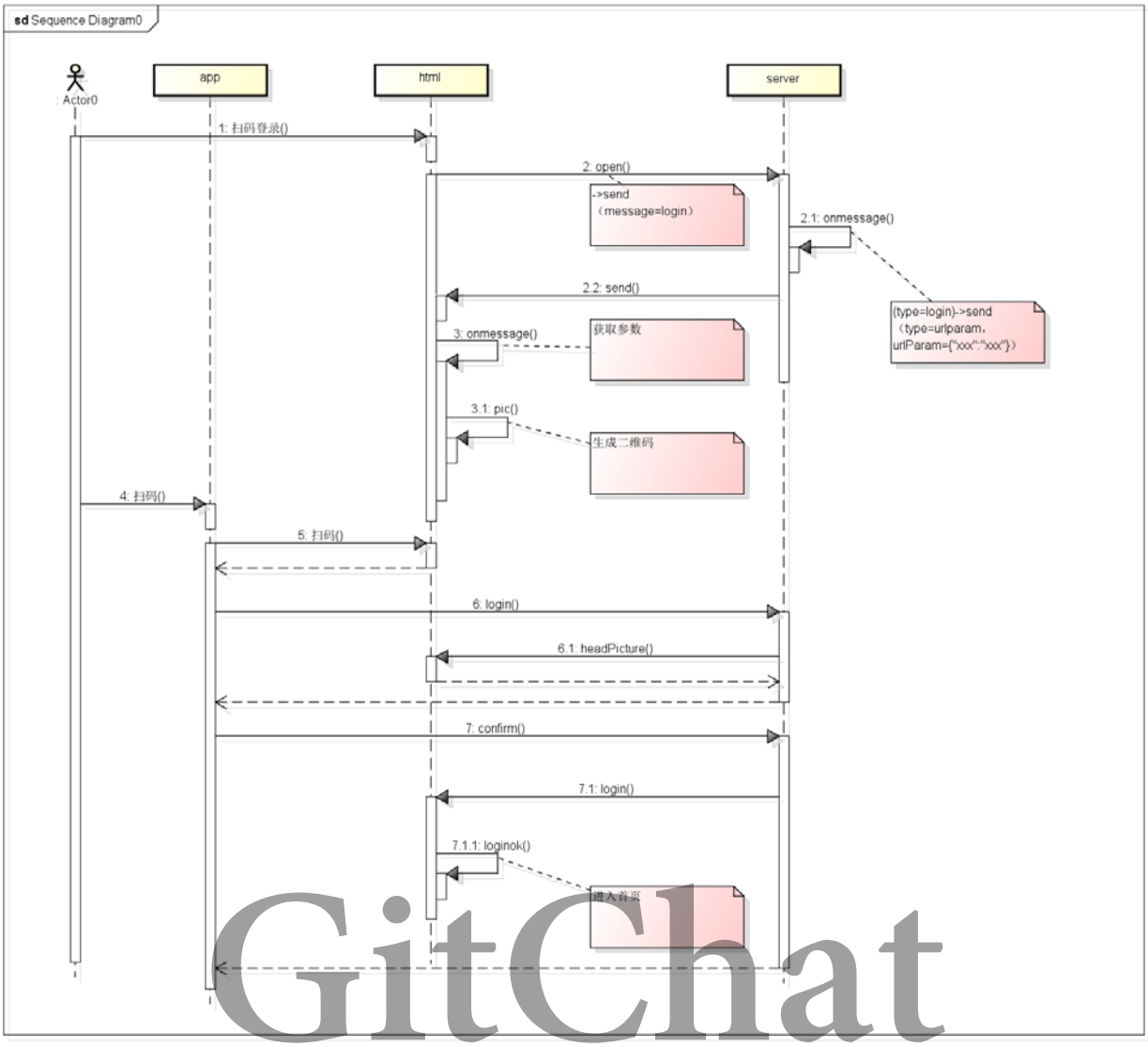
请求登录：

```
    @RequestMapping("login")
    public ResultBean login(@RequestBody WebSocketCondition
webSocketCondition) {
        // 参数包含uuid,token,type
        return userService.login(webSocketCondition);
    }

// userService.login代码片段-------start-----------
// 设定需要返回的数据，将内容发送至相应的session
WebsocketModel websocketModel = new WebsocketModel();
websocketModel.setType("pic");
websocketModel.setMsg(userInfo.getHeadPicture());
// WebSocketUtils.getSession(webSocketCondition.uuid)：通过uuid获
取相应的session，发送信息，说明此次请求已受理，且返回用户头像地址，用于H端显
示用户头像，等待用户确认登录
WebSocketUtils.sendMessage(WebSocketUtils.getSession(webSocketCon
dition.uuid), websocketModel);
// userService.login代码片段-------end----------------
```

确认登录：

```
WebsocketModel websocketModel = new WebsocketModel();
// 获取用户信息
UserInfo userInfo = userDao.get2(UserInfo.class,
SecurityUtils.getPrincipal().getUserId());
```



websocket运用场景

考虑到webSocket的实时性这个特点，就可以运用到各行各业。
看看以下场景是否你考虑到的呢？

1. **即时通讯**：多媒体聊天，你可以使用该技术开个聊天室，聊个火热。可以单独2人

OK，相信大家对webSocket也有了一定的了解了。将功能加入到你需要的地方吧。