## WebSocket 被哪些浏览器所支持



# ESFramework开发手册 -- WebSocket版

 ESFramework的WebSocket客户端引擎的相关API与PC版几乎是一致的，所以，如果熟悉PC版的API，那么上手WebSocket API就很容易了。

## 第一章 发送和处理消息

　 使用通信框架最基础的需求就是收发信息，ESFramework(WebSocket)底层已经为我们封装好了所有与信息收发相关的操作， 当RapidPassiveEngine被new出来以后，RapidPassiveEngine对象就实例化了CustomizeOutter 属性，在对象的initialize函数被调用后，即可调用CustomizeOutter 里面的相关函数来发送消息，以及实现customizeHandler后就可以处理收到的信息。

### 1.客户端发送信息

　　客户端可以发送信息给服务端，也可以发送信息给其他在线用户。CustomizeOutter 里的相关函数说明如下：



发送消息有几种方式

 send(informationType, infobytes, targetUserID)，当调用方省略targetUserID参数时，表示此消息发往服务端

 sendCertainly(informationType, infobytes, callbackFn, targetUserID)，当调用方省略targetUserID参数时，表示此消息发往服务端，当目标方有应答时会调用callbackFn，callbackFn方法签名为function (data){}

 query(informationType, infobytes, callbackFn, targetUserID)，当调用方省略targetUserID参数时，表示此消息发往服务端，当目标方有应答时会调用callbackFn，callbackFn方法签名为function (data){}，和sendCertainly的差别在于，callbackFn的第一个参数，前者无数据，后者有数据

 sendBlob(informationType, blobContent, fragmentSize, targetUserID)，当调用方省略targetUserID参数时，表示此消息发往服务端

### 2.处理信息

客户端可以收到来自其它客户端或服务端的信息、大数据块、以及同步调用。服务端也可以收到来自客户端的信息（转发的信息除外）及同步调用。那么，我们如何处理这些接收到的信息了？

无论是服务端，还是客户端，都只要实现customizeHandler接口即可。



（1）在实现这个接口之后, 只需要实例化这个具体的实现类, 在RapidPassiveEngine的对象被创建后，调用initialize方法时传入即可自动处理接收到的消息

（2）凡是sourceUserID参数为null的，都表示被处理的信息是来自服务端的；否则，表示被处理的信息是由其它在线客户端发出的。

（3）initialize的参数为json格式，参考实现如下

 {

 serverIP: '192.168.5.38',//服务器IP

 serverPort: '4540',//服务器端口

 userID: '10001',//登录用户名

 logonPassword: hex\_md5('1'),//md5后的密码

 heartBeat: 5000,//心跳间隔时间(单位为ms)

 maxLengthOfUserID:11,//设置用户名最大长度

 customizeHandler: new CustomizeHandler(),//用户自定义消息处理器

 loginResutCallBack: function (loginResult) {}//登录结果回调

}

## 第二章 好友与组

(此接口在Esframework6.0中已被IContaOutter取代,为兼容老版本,此接口仍作保留)

大部分分布式通信系统中，除了客户端与服务器进行通信外，都还会涉及到客户端之间相互通信、以及需要将客户端进行分组的功能，或者是类似这方面的需求。ESFramework对这一常见的任务内置了强大的支持，包括从客户端到服务端、一直到ESPlatform群集。在设计时，我们就考虑到了如何对常见的好友通信与组广播通信进行最大的支持，以期让ESFramework的使用者非常容易的就能够使用这些功能。

 在ESFramework中，好友与组成员并不仅仅是指用户（某人），而是指任何一个运行的客户端实例。只要两个客户端实例之间需要频繁相互通信，那么它们就可以建立好友关系（friend）。如果需要在某些特定的客户端实例间进行广播通信，那么这些实例就可以被划分到同一个组，成为组友（groupmate）。如果是使用ESFramework开发类似IM的系统，那么这两种关系就更明显，它们就类似QQ的好友与群。

### 好友关系

当RapidPassiveEngine被new出来以后，RapidPassiveEngine对象就实例化了FriendsOutter属性，在对象的initialize函数被调用后，即可调用FriendsOutter里面的相关函数使用好友关系功能。



（1）用户上/下线时，通知其好友。当用户上线或下线时，框架会触发friendListener接口的FriendConnected或FriendOffline事件以通知所有的在线好友。

（2）客户端可以通过FriendsOutter接口的GetFriends方法和GetAllOnlineFriends方法来获取所有好友以及所有在线的好友列表。

 有了这两组特性的支持，每个运行的客户端实例，在其运行的整个生命周期中，都可以清楚地知道每个好友的在线状态。在具体项目中，我们可以这么做，当某个客户端登陆成功后，就获取所有好友列表和所有的在线好友列表，并预定IFriendsOutter的FriendConnected和FriendOffline事件，然后在运行的过程中，当FriendConnected和FriendOffline事件触发时，就修改对应好友的状态。这样就保证我们的客户端可以实时地知道每个好友是否在线。

### 组友关系

 当RapidPassiveEngine被new出来以后，RapidPassiveEngine对象就实例化了GroupOutter属性，在对象的initialize函数被调用后，即可调用GroupOutter里面的相关函数使用组关系功能。



（1）GetGroupMembers 会返回某个组的所有成员，并将在线成员与不在线成员区分开来。

（2）当用户上线或下线时，框架会回调GroupOutter 接口的GroupmateConnected或GroupmateOffline事件以通知所有的在线组友。

（3）可以通过Broadcast方法向任何一个组发送广播，目标组的每个在线成员都将会通过GroupOutter 的BroadcastReceived事件来获得广播内容。

## 第三章 联系人

此接口在Esframework6.0中用以取代 FriendOutter以及GroupOutter的功能,实际上就是合并了上述两个接口的功能,对方法的命名稍有变动



1. 用户上/下线时，通知相关联系人。当用户上线或下线时，框架会触发ContactsOutter接口的contactsConnected或contactsOffline事件以通知所有的在线好友。

（2）客户端可以通过ContactsOutter接口的getContacts方法和getAllOnlineContacts方法来获取所有好友以及所有在线的好友列表,有了这组特性的支持，每个运行的客户端实例，在其运行的整个生命周期中，都可以清楚地知道每个好友的在线状态。在具体项目中，我们可以这么做，当某个客户端登陆成功后，就获取所有好友列表和所有的在线好友列表，并预定ContactsOutter的事件，然后在运行的过程中，当contactsConnected和contactsOffline事件触发时，就修改对应好友的状态。这样就保证我们的客户端可以实时地知道每个好友是否在线。

（3）GetGroupMembers 会返回某个组的所有成员，并将在线成员与不在线成员区分开来。

（4）可以通过Broadcast方法向任何一个组发送广播，目标组的每个在线成员都将会通过ContactsOutter的BroadcastReceived事件来获得广播内容。

## 第四章 线用户管理、基础功能及状态通知

 在解决了发送信息和处理信息之后，还有一些基础功能是很多分布式通信系统都需要用到的，比如，查询某个用户是否在线、获取在线用户列表、自己掉线时得到通知，等等。BasicOutter，FriendsOutter，GroupOutter，为我们解决了这些基础问题。

### 客户端



状态改变事件通知

首先，我们看看BasicOutter暴露的两个事件：

beingKickedOut 当自己被踢出时将触发该事件。

beingPushedOut 发生于当服务端将重登陆模式设置为ReplaceOld时，并且同名用户的成功登录，将会把老的在线用户挤掉而导致其下线。关于重登陆模式的更多内容可以参见重登陆模式。

### 基础API

接下来，我们简单看看BasicOutter的几个方法。

getAllOnlineUsers用于获取所有在线用户，通常该方法仅仅用于demo，因为在正式的系统中，在线用户数可能是非常巨大的，这将导致GetAllOnlineUsers的返回消息非常大，甚至可能超过框架的最大消息尺寸的限制。

ping方法，用于获取当前客户端到服务端或到另一个在线客户端的消息来回的耗时，由于其是在应用层来模拟类似ICMP的ping，所以这个方法返回的值通常比ICMP的ping大一些。尽管如此，在一些应用中，该Ping的结果还是有一些参考价值的。

有时，我们需要命令服务器将一些恶意的用户从服务端踢出（断开其连接），那么就可以调用kickOut方法，被踢出的客户端将会触发上述的beingKickedOut事件。

sendHeartBeatMessage方法用于向服务器发送心跳消息。如果我们使用的是Rapid引擎，那么框架会自动发送心跳消息，所以，我们通常不需要手动调用该方法。关于心跳消息的更多内容可以参见心跳机制。

### 3.TCP连接状态

 BasicOutter提供了一部分基础功能，还有另一部分很重要的基础功能需要涉及到客户端的Rapid引擎，我们在这里也一并介绍一下。客户端如何知道自己与服务器的TCP连接的状态及其变化了？RapidPassiveEngine的几个属性来获取这些信息。



其中，RapidPassiveEngine 提供的事件以及事件回调方法如下

RapidPassiveEngine.addListener('error', function (error) {});

RapidPassiveEngine.addListener('close', function () {});

RapidPassiveEngine.addListener('open', function () {});

RapidPassiveEngine.addListener('engineMessageReceived', function (senderUID, informationType, bytes, tag) {});

## 第五章 文件传输

### 1. 传输流程

websocket暂不支持接收文件

可以用下图表示：

       

（1） 由发送方发起传送文件的请求。

（2） 接收方回复同意或者拒绝接收文件。如果拒收，则流程结束；否则进入下一步。

（3） 发送方发送文件数据，接收方接收文件数据。

（4） 如果文件传送过程中，接收方或发送方掉线或者取消文件传送，则文件传送被中断，流程结束。如果文件传送过程一直正常，则到最后完成文件的传送。

           有几点需要说明一下：

（1） 发送方可以是客户端，也可以是服务器；接收方也是如此。但无论发送方和接收方的类别如何，它们都遵守这一文件传送流程。

（2） 进行文件传送的线程是由框架自动控制的，只要发送方收到了接收方同意接收的回复，框架就会自动发送文件数据包；而接收方也会自动处理接收到的文件数据包。

（3） 发送方或接收方都可随时取消正在传送的文件。

（4） 当文件传送被中断或完成时，发送方和接收方都会有相应的事件通知。

### 2. 传输接口

客户端可以发送文件给服务端，也可以发送信息给其他在线用户。FileOutter里的相关函数说明如下：



1. 其中beginSendFile函数中的file参数，由h5中 标签 input type="file" 获取，具体使用可参考demo