优客服多渠道客服系统

（UCKeFu）

通

信

功

能

优客服

二〇一七年三月

目录

第一章 客服功能介绍

1.1 优客服功能

1.1.1优客服功能组成

优客服，是一个多渠道融合的客户支持服务平台，包含 WebIM，微信，电话，邮件，短信等接入渠道，在渠道接入方面，优客服主要包含以下几种：

WebIM：优客服提供 WebIM 功能，在线坐席能够通过工作台操作界面，接收来自 WebIM 的咨询请求，优客服通过整合多个渠道来源，让坐席在同一个工作界面上处理来自 PC 端、移动端、微信端，微博等渠道的服务请求。

社交媒体： 接入微信和微博渠道，将社交媒体渠道的的咨询请求接入进入优客服 坐席工作平台，让客服统一响应和受理

邮件、短信： 多种邮件处理方式，能够将邮箱的消息转为坐席的待处理任务，可以将待处理任务或邮件转为工单

电话语音接入：呼叫中心功能，能够接入电话呼叫中心功能。

1.1.2 实时通信组件部分功能

实时消息的推送 ,PC 端的推送技术可以使用 socket 建立一个长连接来实现。传统的 web 服务都是客户端发出请求，服务端给出响应。但是现在直观的要求是允许特定时间内在没有客户端发起请求的情况下服务端主动推送消息到客户端。

实现 webim 方法有很多，最关键的问题是保持和服务端的连接。如何保障会话的即时性和连通性。常用的有 poll, long poll, comet 等。

有哪些可以实现 web 消息推送的技术：

不断地轮询（俗称“拉”，polling）是获取实时消息的一个手段：Ajax 隔一段时间（通常使用 JavaScript 的 setTimeout 函数）就去服务器查询是否有改变，从而进行增量式的更新。但是间隔多长时间去查询成了问题，因为性能和即时性造成了严重的反比关系。间隔太短，连续不断的请求会冲垮服务器，间隔太长，务器上的新数据就需要越多的时间才能到达客户机。

优点：服务端逻辑简单；

缺点：其中大多数请求可能是无效请求，在大量用户轮询很频繁的情况下对服务器的压力很大；

应用：并发用户量少，而且要求消息的实时性不高，一般很少采用；

长轮询技术（long-polling）：客户端向服务器发送 Ajax 请求，服务器接到请求后 hold 住连接，直到有新消息或超时（设置）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：实时性高，无消息的情况下不会进行频繁的请求；

缺点：服务器维持着连接期间会消耗资源；

基于 Iframe 及 htmlfile 的流（streaming）方式：iframe 流方式是在页面中插入一个隐藏的 iframe，利用其 src 属性在服务器和客户端之间创建一条长链接，服务器向 iframe 传输数据（通常是 HTML，内有负责插入信息的 javascript），来实时更新页面。

优点：消息能够实时到达；

缺点：服务器维持着长连接期会消耗资源；

插件提供 socket 方式：比如利用 Flash XMLSocket，Java Applet 套接口，Activex 包装的 socket。

优点：原生 socket 的支持，和 PC 端和移动端的实现方式相似；

缺点：浏览器端需要装相应的插件；

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。

优点：更好的节省服务器资源和带宽并达到实时通讯；

缺点：目前还未广泛普及，浏览器支持不好，IE9 及以下的浏览器不能很好的支持；

综上，考虑到浏览器兼容性和性能问题，采用长轮询（long-polling）是一种比较好的方式。

netty-socketio 是一个开源的 Socket.io 服务器端的一个 java 的实现， 它基于 Netty 框架。

1.1.3 NettySocketIO 介绍

NettySocketIO 是一个开源的通信服务器的 java 的实现， 它基于 Netty 框架。可应用于服务端主动推送消息到客户端等场景，和 WebSocket 是功能类似，只 NettySocketIO 可支持大部分主流的浏览器。

项目地址为： https://github.com/mrniko/netty-socketio。

Socket.IO 除了支持 WebSocket 通讯协议外，还支持许多种轮询（Polling）机制以及其它实时通信方式，并封装成了通用的接口，并且在服务端实现了这些实时机制的相应代码。Socket.IO 实现的 Polling 通信机制包括 Adobe Flash Socket、AJAX 长轮询、AJAX multipart streaming、持久 Iframe、JSONP 轮询等。Socket.IO 能够根据浏览器对通讯机制的支持情况自动地选择最佳的方式来实现网络实时应用。

1.1.4 NameSpace 和 Room 的概念

NameSpace 和 Room 的概念其实用来同一个服务端 Socket 多路复用的。NameSpace，Room 和 Socket 的关系如下。

Socket 会属于某一个 Room，如果没有指定，那么会有一个默认的 Room。这个 Room 又会属于某个 NameSpace，如果没有指定，那么就是默认的 NameSpace/.

最后 SocketIO 有用所有的 NameSpace。

客户端连接时指定自己属于哪个 NameSpace,

io.connect(http://localhost/namespace)。 服务端看到 NameSpace 就会把这个 socket 加入指定的 NameSpace。

如果客户端没有具体指定哪个 room，则服务端会放入默认 room 中，或者服务端通过代码 socket.join("agent") 放入 agent 的 Room 中。

有了这个概念之后，就比较好理解 Socket.IO 是如何广播的，广播的时候是以 NameSpace 为单位的 ,如果只想广播给某个 Room ,那就需要另外指定 Room 的名字。

socketio.send("send to the clients which belong to default namespace(/)");

这个调用没有指定 NameSpace 和 Room，那么这个广播的对象就是 广播给默认 NameSpace/ 和默认 Room。 如果你的客户端连接到服务器的 path 是 http://localhost/agent,

客户端是属于 agent NameSpace，那么这个客户端就收不到这个消息。只有那些 path 是 http://localhost 的才能收到。

socket.broadcast.emit('message', "send to the clients which belong to namespace(socket belong to) except sender");

通过 socket 广播时，是广播给这个 socket 所属的 namespace 里的所有客户端。只有跟 socket 同一个 namespace 里的客户端才能收到数据。

socket.broadcast.in('chat).emit('message', "send to the clients which belong to namespace(socket belong to) except sender");

广播给跟 socket 同一个 NameSpace 下面的，名字为 chat 的 room 里的除自己以外的客户端。

socketio.of('/private').send("send to all the clients which belong to namespace(priavte)");

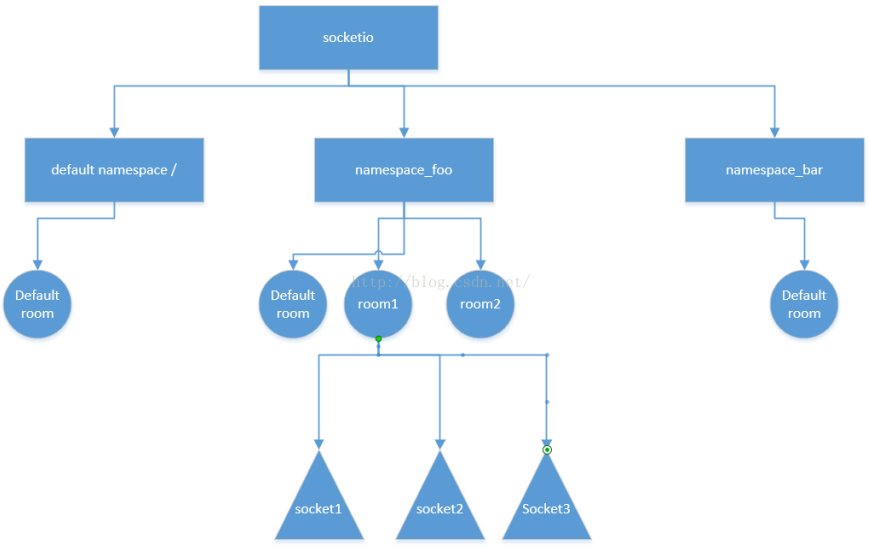
发给 private namespace 里的所有客户端。

socketio.of('/private').in('chat').send("send to all the clients in the chat room which belong to namespace(private)");

发给 private namespace 里面的 chat room 的所有客户端。

记忆这些函数也比较容易 ，如果是 socket 开头的，那么 namespace 已经指定，只能修改 room。如果是 socketio 开头的，那么可以指定 namesapce 和 room。

如果没有任何指定，那就是默认的 NameSpace 和默认的 Room。



1.2 在线客服功能

1.2.1 在线客服

在线客服系统是一种网页版即时通讯软件的统称。相比较其他即时通讯软件（如 QQ、MSN 等），它实现和网站的无缝结合，为网站提供和访客对话的平台；网站访客无需安装任何软件，即可通过网页进行对话。在线客服系统共有的一个特点是：网站的所有者想要使用在线客服系统，必须先向在线客服系统申请一个帐户，然后生成网页标签，就是一段代码，然后把这段代码嵌入到网站网页当中。用申请的帐户登录在线客服系统，就可以开始用了。

优客服通信功能包括两个组成部分，访客端功能和坐席端功能，其中访客端功能实现了访客跟踪、对话、邀请、传送文件等功能。

1.2.2 访客端功能

1、访客对话

访客无需安装任何插件，通过文字、图片、截图等方式与客服交流。

2、主动邀请

访客进入网站时，根据设置的参数系统自动发对话出邀请。

3、发送图片

工作平台和访客端双向图片发送传输功能，并支持常用图片的管理。

5、消息预知

客服可以同步看到访客正在输入未发送的文字。

6、常用语管理

可以对常用语进行预设、归类、编辑、查询等操作。

7、对话转接

客服遇到疑难问题时，可以转接给相应同事，寻求帮助，提高服务水平。

8、指定客服

不同页面的咨询可以指定由相应的客服或者部门来接待。

9、聊天记录

记录中心提供聊天记录的存储和查询。

10、外观自定义

邀请框、图标、访客端界面均可实现自定义设置。

11、访客自动识别

二次访问的访客，可显示上次对话记录以及客户信息。

1.2.3 坐席端功能

1、客服接待

邀请对话：以邀请框的方式出现在访客当前正在浏览页面。

访客分配（ACD）：多个坐席同时接待客户的时候，能将客户通过分配算法分配给合适的坐席。

客服转接：客服可将对话转接给其他在线客服接待。

消息预知：同步看到访客正在输入的文字信息。

查看访客轨迹：访客轨迹包括访客当次访问记录以及历史访问轨迹两部分。

查看访客信息：查看访客基本信息、联系人、服务信息等，工单等信息。

2、客服监管

对话实时监控：坐席主管可实时监控客服聊天内容，支持加入和强接对话。

客服 KPI 考核：包括客服在线时长、接待数量、坐席评分、接待质量等 KPI 统计

3、风格设置

样式自定义：图标、邀请框、访客端界面可自定义。

智能 ACD 路由：包含平均轮询、空闲坐席优先、历史接待坐席优先等多种接待规则。

对话提示语设置：包括坐席忙、结束对话、接通、离开等各种提示语设置。

4、客户管理

分配客户：自动分配客户资源，与在线客服完美整合。

自定义联系人属性：联系人属性自定义，满足不同行业需求。

标签管理：自定义访客标签，区分联系人类别与销售阶段。

联系记录：详细记录客户联系情况，系统自动提醒待联系任务。

工单管理：包括工单管理，记录客户工单记录，可在客服端方便查看。

数据分析：包括访问统计、工单统计、坐席绩效统计等维度分析。

第二章 通信组件介绍

2.1 服务配置代码

优客服通信服务配置通过 SpringBoot 启动，服务配置代码位于：

com.ukefu.webim.config.web. IMServerConfiguration，使用的是 SpringBoot 的@ Configuration 功能启动，以下是关键代码说明：

```
@Value("${uk.im.server.host}")
private String host;

@Value("${uk.im.server.port}")
private Integer port;

@Value("${web.upload-path}")
private String path;

private SocketIOServer server ;

@Bean(name="webimport")
public Integer getWebIMPort() {
UKDataContext.setWebIMPort(port);
return port;
}
```

1、uk.im.server.host

是 IM 服务器的访问地址，主要作用用于配置 NettySocketIO 的启动参数，在解决 CROSS 问题的时候提供 HOST 标识。

2、uk.im.server.port

是 IM 服务器的访问端口，主要作用用于启动 IM 服务监听端口

3、web.upload-path

是优客服系统的文件存储位置 ,单服务器使用的时候 ,配置为本地存储路径；如果以集群部署使用，建议采用 SAN 路径。

2.2 服务注册代码

服务注册代码包含以下两段代码，1、注册 SocketIOServer，2、启动 SocketIOServer：

```
@Bean
    public SocketIOServer socketIOServer() throws NoSuchAlgorithmException,
IOException
    {
         Configuration config = new Configuration();
//               config.setHostname("localhost");
                 config.setPort(port);
//               config.setSocketConfig(new SocketConfig());
//               config.setOrigin("http://im.uckefu.com");
                 config.setExceptionListener(new UCKeFuExceptionListener());
                 File sslFile = new File(path , "ssl/https.properties") ;
       if(sslFile.exists()){
        Properties sslProperties = new Properties();
        FileInputStream in = new FileInputStream(sslFile);
        sslProperties.load(in);
        in.close();
        if(!StringUtils.isBlank(sslProperties.getProperty("key-store"))
&& !StringUtils.isBlank(sslProperties.getProperty("key-store-password"))){

        config.setKeyStorePassword(UKTools.decryption(sslProperties.getProp
erty("key-store-password")));
            InputStream stream = new FileInputStream(new File(path ,
"ssl/"+sslProperties.getProperty("key-store")));
            config.setKeyStore(stream);
        }
       }


//          config.setSSLProtocol("https");

                 config.setWorkerThreads(100);
//               config.setStoreFactory(new HazelcastStoreFactory());
```

```
                config.setAuthorizationListener(new AuthorizationListener()
{
                        public boolean isAuthorized(HandshakeData data) {
                                return true;
                        }
                });

        return server = new SocketIOServer(config);
    }
```

注：如果系统配置了 SSL 服务证书，在启动 IM 服务端的时候，会加载 SSL 证书。

```
@Bean
public SpringAnnotationScanner springAnnotationScanner(SocketIOServer
socketServer) {
        return new SpringAnnotationScanner(socketServer);
}
```

2.3 服务启动代码

服务端启动代码是通过 SpringBoot 的@Component 注册：

```
@Autowired
public ServerRunner(SocketIOServer server) {
        this.server = server;
        imSocketNameSpace =
server.addNamespace(UKDataContext.NameSpaceEnum.IM.getNamespace())  ;
        agentSocketIONameSpace =
server.addNamespace(UKDataContext.NameSpaceEnum.AGENT.getNamespace()) ;
        entIMSocketIONameSpace =
server.addNamespace(UKDataContext.NameSpaceEnum.ENTIM.getNamespace()) ;
}
```

服务声明了三个 NameSpace，分布用于处理访客对话功能，坐席对话功能和坐席间的对话功能（EntIM）

以下代码分别为 Namespace 创建了事件和消息处理类：

```
@Bean(name="imNamespace")
public SocketIONamespace getIMSocketIONameSpace(SocketIOServer server ){
```

```
        imSocketNameSpace.addListeners(new IMEventHandler(server));
        return imSocketNameSpace  ;
}

@Bean(name="agentNamespace")
public SocketIONamespace getAgentSocketIONameSpace(SocketIOServer server){
        agentSocketIONameSpace.addListeners(new AgentEventHandler(server));
        return agentSocketIONameSpace;
}

@Bean(name="entimNamespace")
public SocketIONamespace getEntIMSocketIONameSpace(SocketIOServer server){
        entIMSocketIONameSpace.addListeners(new EntIMEventHandler(server));
        return entIMSocketIONameSpace;
}
```

事件和消息的处理类主要代码如下：

```
//消息接收入口，当接收到消息后，查找发送目标客户端，并且向该客户端发送消息，且给自己发送消息
@OnEvent(value = "service")
public void onEvent(SocketIOClient client, AckRequest request,
AgentServiceMessage data)
{

}

//消息接收入口，当接收到消息后，查找发送目标客户端，并且向该客户端发送消息，且给自己发送消息
@OnEvent(value = "status")
public void onEvent(SocketIOClient client, AckRequest request,
AgentStatusMessage data)
{

}
```

注册的消息类型有以下几种：

1、@OnConnect

客户端建立连接的事件处理接口

2、@OnDisconnect

客户端断开连接的事件处理接口

3、@OnEvent

客户端触发的消息事件处理接口，优客服系统中定义了以下几种事件消息：

1、@OnEvent(value = "service")

消息接收入口，当接收到消息后，查找发送目标客户端，并且向该客户端发送消息，且给自己发送消息

2、@OnEvent(value = "status")

状态变更事件，例如，新用户接入、用户离开、统计坐席的当前状态信息等

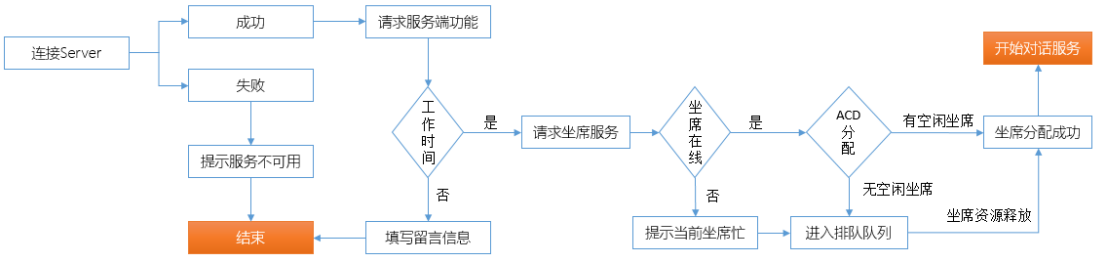
3、@OnEvent(value = "message")

介绍到访客或坐席发送的消息处理接口，完成消息的解析、组装、路由等功能

第三章 客户端通信介绍

3.1 客户端工作流程

客户端功能包含三个页面，访客入口按钮、坐席对话、留言页面，其中，坐席对话页面的处理流程如下：



处理代码如下：

```
var socket =
io.connect('${schema!'http'}://'+hostname+':${port}/im/user?userid=${userid
!''}&orgi=${orgi!''}&session=${sessionid!''}&appid=${appid!''}&osname=${(os
name!'')?url}&browser=${(browser!'')?url}<#if
skill??>&skill=${skill}</#if><#if agent??>&agent=${agent}</#if>');
```

通过 SocketIO.js 代码连接服务端，连接的 NameSpace 是 user，同时，传入连接的参数，各个参数及其功能说明如下：

userid：当前访客的用户标识，通过浏览器指纹识别

orgi：租户标识

session：会话标识

appid：网站识别代码，能够支持多个网站接入客服系统

osname：当前访问客户端的操作系统名称

browser：当前访问客户端的浏览器名称和版本信息

skill：接入的技能组，默认为空，表示自动分配

agent：请求的人工坐席，默认为空，表示自动分配

3.2 客户端消息处理

```
socket.on('disconnect',function() {
       output('<span id="connect-message">连接坐席失败，在线咨询服务不可用
</span>' , 'message connect-message');
});
```

链接失败的消息，服务端关闭了对话服务器

```
socket.on("agentstatus",function(data){
        document.getElementById('connect-message').innerHTML=data.message;
})
```

用于接收服务端推送的坐席分配状态消息，例如：坐席忙、坐席分配成功

```
socket.on("status",function(data){
                output('<span
id="connect-message">'+data.message+'</span>' , 'message connect-message');
                if(data.messageType == "end"){
                        service_end = true ;
                        editor.readonly();
                }
    })
```

接收客服系统状态消息，例如开始和结束坐席对话

```
    socket.on('message', function(data) {
           var
chat=document.getElementsByClassName('chatting-left').innerText;
           chat = data.message;
           if(data.messageType == "image"){
```

```
                chat = "<img src='"+data.message+"'
class='ukefu-media-image'>" ;
          }
                if(data.calltype == "in"){
                        output('<div class="chat-right"> <img
class="user-img" src="/im/img/user.png" alt=""><div
class="chat-message"><label class="time">'+data.createtime+'</label><label
class="user">'+data.nickName+'</label> </div><div class="chatting-right"><i
class="arrow arrow${inviteData.consult_dialog_color!''}"></i><div
class="chat-content
theme${inviteData.consult_dialog_color!''}">'+chat+'</div></div>' ,
"chat-block");
                }else if(data.calltype == "out"){
                        output('<div class="chat-left"> <img
class="user-img" src="<#if inviteData?? &&
inviteData.consult_dialog_headimg??>/res/image.html?id=${inviteData.consult
_dialog_headimg?url}<#else>/images/agent.png</#if>" alt=""><div
class="chat-message"><label  class="user">'+data.nickName+'</label><label
class="time">'+data.createtime+'</label> </div><div class="chatting-left"><i
class="arrow"></i><div class="chat-content">'+chat+'</div></div>' ,
"chat-block");
                }
       });
```

接收坐席发送的消息和当前访客发送给坐席的消息，支持文本、图片和表情消息。

3.3 客户端发送消息

访客和坐席的对话连接建立之后，访客可以发送消息给坐席，发送消息的代码如下：

```
   function sendMessage() {
        editor.sync();
        var count = editor.count("text");
        if(count>0 && service_end == false){
        var message = document.getElementById('message').value;
        if(message!= ""){
                        socket.emit('message', {
```

```
                                    appid : "${appid!''}",
                                    userid:"${userid!''}",
                                    type:"message" ,
                                    session:"${sessionid!''}",
                                    orgi:"${orgi!''}",
                                    message : message
                    });
            }
        }else if(service_end == true){
            alert("坐席已断开和您的对话");
        }
        editor.html('');
```

主要的代码为 socket.emit('message', {}) ;