Artículo
Ricardo Paiva · 6 dic, 2019 Lectura de 12 min

Tutorial de WebSockets

¡Hola Comunidad!

La mayor parte de las comunicaciones servidor-cliente en la web se basan en una estructura de solicitud y respuesta. El cliente envía una solicitud al servidor y el servidor responde a esta solicitud. El protocolo WebSocket ofrece un canal bidireccional de comunicación entre un servidor y un cliente, lo que permite a los servidores enviar mensajes a los clientes sin antes haber recibido una solicitud. Por más información sobre el protocolo WebSocket y su implementación en InterSystems IRIS, vea los siguientes enlaces:

- WebSocket protocol
- WebSockets in InterSystems IRIS documentation

Este tutorial es una actualización del tutorial rápido "Asynchronous Websockets -- a quick tutorial" para Caché 2016.2+ e InterSystems IRIS 2018.1+.

Funcionamiento asíncrono vs síncrono

En InterSystems IRIS, una conexión WebSocket puede implementarse de forma síncrona o asíncrona. El tipo de funcionamiento de la conexión WebSocket entre el cliente y el servidor está determinada por la propiedad “ SharedConnection ” de la clase %CSP.WebSocket.

- SharedConnection=1 : funcionamiento asíncrono
- SharedConnection=0: funcionamiento síncrono

Una conexión WebSocket entre un cliente y un servidor basado en InterSystems IRIS incluye una conexión entre la instancia de InterSystems IRIS y la Puerta de enlace web. En la forma de funcionamiento síncrona de WebSocket, la conexión usa un canal privado. En el funcionamiento asíncrono de WebSocket, un grupo de clientes WebSocket comparten un conjunto de conexiones entre la instancia de InterSystems IRIS y la Puerta de enlace web. La ventaja de la implementación asíncrona de WebSockets se hace más notoria cuando uno tiene varios clientes que se conectan al mismo servidor, ya que esta implementación no requiere que cada cliente cuente con una conexión exclusiva entre la Puerta de enlace web y la instancia de InterSystems IRIS.
En este tutorial implementaré WebSockets de forma asíncrona. Por lo tanto, todas las ventanas de chat abiertas comparten un conjunto de conexiones entre la Puerta de enlace web y la instancia de InterSystems IRIS que aloja la clase de servidor de WebSocket.

Resumen de la aplicación de chat

El "hola mundo" de WebSockets es una aplicación de chat en la que el usuario puede enviar mensajes que se transmiten a todos los usuarios que hayan iniciado sesión en la aplicación. En este tutorial, los componentes de la aplicación de chat incluyen:

- Servidor: implementado en una clase que extiende a %CSP.WebSocket
- Cliente: implementado por una página CSP

La implementación de esta aplicación de chat logrará lo siguiente:

- Los usuarios podrán transmitir mensajes a todas las ventanas de chat abiertas a la vez (broadcast)
- Los usuarios conectados aparecerán en la lista de usuarios conectados ("Online Users") de todas las ventanas de chat abiertas
- Los usuarios podrán cambiar su nombre de usuario con solo escribir un mensaje que comience con la palabra clave "alias" y este mensaje no se transmitirá, pero actualizará la lista de usuarios conectados
- Cuando los usuarios cierren su ventana de chat, se eliminarán de la lista de usuarios conectados

Para ver el código fuente de la aplicación de chat, visite este repositorio de GitHub.

El cliente

El lado cliente de nuestra aplicación de chat lo implementa una página CSP que contiene la información de estilo de la ventana de chat, la declaración de la conexión WebSocket, los eventos y métodos de WebSocket que manejan la comunicación hacia y desde el servidor y las funciones de ayuda que empaquetan los mensajes enviados al servidor y procesan los mensajes entrantes.
Primero analizaremos cómo la aplicación inicia la conexión WebSocket usando una biblioteca de WebSocket para Javascript.

```
ws = new WebSocket(((window.location.protocol === "https:")? "wss":"ws:")
                   + "//"+ window.location.host + "/csp/user/Chat.Server.cls");
```

new crea una nueva instancia de la clase WebSocket. Esto abre una conexión de WebSocket al servidor usando el protocolo "wss" (indica el uso de TLS para el canal de comunicación de WebSocket) o el "ws". El servidor queda determinado por el número de puerto de servidor web y el nombre de host de la instancia que define la clase `Chat.Server` (esta información está incluida en la variable `window.location.host`. El nombre de nuestra clase (Chat.Server.cls) está incluido en la URI de apertura del WebSocket como una solicitud `GET` del recurso en el servidor.
El evento `ws.onopen` se dispara cuando la conexión de WebSocket se establece con éxito, y pasa del estado *connecting* (conectando) a *open* (abierta).

```
ws.onopen = function(event){
        document.getElementById("headline").innerHTML = "CHAT - CONNECTED";
    };
```

Este evento actualiza el encabezado de la ventana de chat de forma de indicar que el cliente y el servidor están conectados.

Envío de mensajes

La acción de que un usuario envíe un mensaje dispara la función `send` (enviar). Esta función sirve como envoltorio del método `ws.send`, que contiene las mecánicas para enviar el mensaje al servidor sobre una conexión WebSocket.

```
function send() {
    var line=$("#inputline").val();
    if (line.substr(0,5)=="alias"){
        alias=line.split(" ")[1];
        if (alias==""){
            alias="default";
        }
        var data = {}
        data.User = alias
        ws.send(JSON.stringify(data));
        } else {
        var msg=btoa(line);
```

```
        var data={};
        data.Message=msg;
        data.Author=alias;
        if (ws && msg!="") {
            ws.send(JSON.stringify(data));
        }
    }
    $("#inputline").val("");
}
```

send empaqueta la información a enviar al servidor en un objeto JSON, para lo cual define pares de clave/valor de acuerdo con el tipo de información a enviar (actualización de alias o mensaje general). `btoa` traduce el contenido de un mensaje general a una cadena de texto ASCII con cifrado de base 64.

Recepción de mensajes

Cuando el cliente recibe un mensaje del servidor, se dispara el evento `ws.onmessage`.

```
ws.onmessage = function(event) {
    var d=JSON.parse(event.data);
    if (d.Type=="Chat") {
        $("#chat").append(wrapmessage(d));
            $("#chatdiv").animate({ scrollTop: $('#chatdiv').prop("scrollHeight")}, 1
000);
    } else if(d.Type=="userlist") {
        var ul = document.getElementById("userlist");
        while(ul.firstChild){ul.removeChild(ul.firstChild)};
        $("#userlist").append(wrapuser(d.Users));
    } else if(d.Type=="Status"){
        document.getElementById("headline").innerHTML = "CHAT - connected - "+d.WSID;
    }
};
```

Dependiendo del tipo de mensaje recibido por el cliente (“ Chat ” , “ userlist ” o “ status ”), el mensaje llama a `wrapmessage` o `wrapuser` para poblar las secciones apropiadas de la ventana de chat con los datos entrantes. Si el mensaje entrante es una actualización de estado, el encabezado de estado de la ventana de chat se actualiza con el ID de WebSocket, lo que define la conexión WebSocket bidireccional asociada con la ventana de chat.

Componentes adicionales del cliente

Un error en la comunicación entre el cliente y el servidor dispara el método `onerror` de WebSocket, que emite un alerta para notificarnos del error y actualiza el encabezado de estado de la página.

```
ws.onerror = function(event) {
    document.GetElementById("headline").innerHTML = "CHAT - error";
    alert("Received error");
};
```

Cuando la conexión WebSocket entre el cliente y el servidor se cierra, el método `onclose` se dispara y actualiza el encabezado de estado.

```
ws.onclose = function(event) {
    ws = null;
    document.getElementById("headline").innerHTML = "CHAT - disconnected";
}
```

El servidor

El lado servidor de la aplicación de chat es implementado por la clase `Chat.Server`, que extiende a `%CSP.WebSocket`. Nuestra clase de servidor hereda varias propiedades y métodos de `%CSP.WebSocket`, algunas de las cuales analizaré a continuación. `Chat.Server` también implementa métodos personalizados para procesar mensajes provenientes de, y transmitir mensajes hacia, los clientes.

Antes de iniciar el servidor

`OnPreServer()` se ejecuta antes de crearse el WebSocket y se hereda de la clase `%CSP.WebSocket`.

```
Method OnPreServer() As %Status
{
    set ..SharedConnection=1
    if (..WebSocketID '= ""){
        set ^Chat.WebSocketConnections(..WebSocketID)=""
    } else {
        set ^Chat.Errors($INCREMENT(^Chat.Errors),"no websocketid defined")=$HOROLOG
    }
    Quit $$$OK
}
```

Este método establece en 1 el parámetro de clase `SharedConnection`, lo que indica que nuestra conexión WebSocket será asíncrona y soportada por múltiples procesos que definen las conexiones entre la instancia de InterSystems IRIS y la puerta de enlace web. El parámetro `SharedConnection` solo se puede modificar en `OnPreServer()`. `OnPreServer()` también almacena el ID de WebSocket asociado con el cliente en la global `^Chat.WebSocketConnections`.

El método Server

El método `Server()` contiene el cuerpo de lógica principal ejecutado por el servidor.

```
Method Server() As %Status
{
    do ..StatusUpdate(..WebSocketID)
    for {
        set data=..Read(.size,.sc,1)
        if ($$$ISERR(sc)){
            if ($$$GETERRORCODE(sc)=$$$CSPWebSocketTimeout) {
                //$$$DEBUG("no data")
            }
            if ($$$GETERRORCODE(sc)=$$$CSPWebSocketClosed){
                kill ^Chat.WebSocketConnections(..WebSocketID)
                do ..RemoveUser($g(^Chat.Users(..WebSocketID)))
                kill ^Chat.Users(..WebSocketID)
                quit  // Client closed WebSocket
            }
        } else{
            if data["User"{
                do ..AddUser(data,..WebSocketID)
            } else {
                set mid=$INCREMENT(^Chat.Messages)
                set ^Chat.Messages(mid)=data
```

```
                do ..ProcessMessage(mid)
            }
        }
    }
    Quit $$$OK
}
```

Este método lee los mensajes entrantes desde el cliente (usando el método `Read` de la clase `%CSP.WebSockets`), agrega los objetos JSON recibidos al global `^Chat.Messages` y llama a `ProcessMessage()` para reenviar el mensaje a todos los demás clientes de chat conectados. Cuando un usuario cierra su ventana de chat (lo que finaliza la conexión WebSocket al servidor), la llamada del método `Server()` a `Read` devuelve un código de error que se evalúa en la macro `$$$CSPWebSocketClosed` y el método procede a manejar el cierre de forma acorde.

Procesamiento y distribución de mensajes

`ProcessMessage()` agrega metadatos al mensaje de chat entrante y llama a `SendData()`, al que le pasa el mensaje como parámetro.

```
ClassMethod ProcessMessage(mid As %String)
{
    set msg = ##class(%DynamicObject).%FromJSON($GET(^Chat.Messages(mid)))
    set msg.Type="Chat"
    set msg.Sent=$ZDATETIME($HOROLOG,3)
    do ..SendData(msg)
}
```

`ProcessMessage()` recupera el mensaje con formato JSON desde el global `^Chat.Messages` y lo convierte en un objeto de InterSystems IRIS usando el método `%FromJSON` de la clase `%DynamicObject`. Esto nos permite editar los datos fácilmente antes de reenviar el mensaje a todos los clientes de chat conectados. Agregamos un atributo `Type` (tipo) con el valor "Chat," que el cliente usará para determinar qué hacer con el mensaje entrante. `SendData()` envía el mensaje a todos los demás clientes de chat conectados.

```
ClassMethod SendData(data As %DynamicObject)
{
    set c = ""
    for {
        set c = $order(^Chat.WebSocketConnections(c))
        if c="" Quit
        set ws = ..%New()
        set sc = ws.OpenServer(c)
        if $$$ISERR(sc) { do ..HandleError(c,"open") }
        set sc = ws.Write(data.%ToJSON())
        if $$$ISERR(sc) { do ..HandleError(c,"write") }
    }
}
```

`SendData()` vuelve a convertir el objeto de InterSystems IRIS en una cadena JSON (`data.%ToJSON()`) y hace un envío push del mensaje a todos los clientes del chat. `SendData()` recibe el ID de WebSocket ID asociado a cada conexión cliente-servidor desde el global `^Chat.WebSocketConnections` y usa el ID para abrir una conexión WebSocket mediante el método `OpenServer` de la clase `%CSP.WebSocket`. Para hacer esto podemos usar el método `OpenServer`, porque nuestras conexiones WebSocket son asíncronas: tomamos uno del conjunto compartido existente de procesos InterSystems IRIS-Puerta de enlace web y le asignamos el ID de WebSocket que identifica la conexión del servidor a un cliente de chat específico. Finalmente, el método `Write()` de `%CSP.WebSocket` hace un envío push al cliente de la representación del mensaje como cadena JSON.

Conclusión

Esta aplicación de chat demuestra cómo establecer conexiones WebSocket entre un cliente y un servidor alojado en InterSystems IRIS. Para leer más sobre el protocolo y su implementación en InterSystems IRIS, siga los enlaces que dejé en la introducción.

#Frontend #JavaScript #Node.js #ObjectScript #Tutorial #Web Gateway #Caché #InterSystems IRIS #Open Exchange