Artigo
Lily Taub · Dez. 21, 2020 9min de leitura

Um tutorial sobre WebSockets

Introdução

A maior parte da comunicação servidor-cliente na web é baseada em uma estrutura de solicitação e resposta. O cliente envia uma solicitação ao servidor e o servidor responde a esta solicitação. O protocolo WebSocket fornece um canal bidirecional de comunicação entre um servidor e um cliente, permitindo que os servidores enviem mensagens aos clientes sem primeiro receber uma solicitação. Para obter mais informações sobre o protocolo WebSocket e sua implementação no InterSystems IRIS, consulte os links abaixo.

- Protocolo WebSocket
- WebSockets na documentação do InterSystems IRIS

Este tutorial é uma atualização de"Asynchronous Websockets - um tutorial rápido" para Caché 2016.2+ e InterSystems IRIS 2018.1+.

Operação assíncrona vs síncrona

No InterSystems IRIS, uma conexão WebSocket pode ser implementada de forma síncrona ou assíncrona. O modo como a conexão WebSocket entre o cliente e o servidor opera é determinado pela propriedade "SharedConnection" da classe %CSP.WebSocket.

- SharedConnection=1 : operação assíncrona
- SharedConnection=0: operação síncrona

Uma conexão WebSocket entre um cliente e um servidor hospedado em uma instância do InterSystems IRIS inclui uma conexão entre a instância IRIS e o Web Gateway. Na operação síncrona de WebSocket, a conexão usa um canal privado. Na operação assíncrona de WebSocket, um grupo de clientes WebSocket compartilha um conjunto de conexões entre a instância IRIS e o Web Gateway. A vantagem de uma implementação assíncrona de WebSockets se destaca quando se tem muitos clientes se conectando ao mesmo servidor, pois esta implementação não exige que cada cliente seja tratado por uma conexão exclusiva entre o Web Gateway e a instância IRIS.

Neste tutorial, implementaremos WebSockets de forma assíncrona. Portanto, todas as janelas de bate-papo abertas compartilham um conjunto de conexões entre o Web Gateway e a instância IRIS que hospeda a classe do servidor WebSocket.

Visão geral da aplicação de chat

O "hello world" do WebSockets é uma aplicação de chat em que um usuário pode enviar mensagens que são transmitidas a todos os usuários logados na aplicação. Neste tutorial, os componentes da aplicação de chat incluem:

- Servidor: implementado em uma classe que estende %CSP.WebSocket
- Cliente: implementado por uma página CSP

A implementação desta aplicação de chat irá realizar o seguinte:

- Os usuários podem transmitir mensagens para todas as janelas do chat abertas
- Os usuários on-line aparecerão na lista "Usuários on-line" de todas as janelas de chat abertas
- Os usuários podem alterar seu nome de usuário compondo uma mensagem começando com a palavra-chave "alias" e esta mensagem não será transmitida, mas atualizará a lista de "Usuários On-line"
- Quando os usuários fecham a janela de chat, eles são removidos da lista de "Usuários on-line"

Para ver o código fonte da aplicação de chat, visite esterepositório no GitHub.

O Cliente

O lado do cliente da nossa aplicação de chat é implementado por uma página CSP contendo o estilo para a janela do chat, a declaração da conexão WebSocket, eventos do WebSocket e métodos que tratam da comunicação de e para o servidor, e funções auxiliares que empacotam mensagens enviadas para o servidor e processam as mensagens de entrada.

Primeiro, veremos como a aplicação inicia a conexão WebSocket usando uma biblioteca Javascript WebSocket.

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

new cria uma nova instância da classe WebSocket. Isso abre uma conexão WebSocket com o servidor usando o protocolo "wss" (indica o uso de TLS para o canal de comunicação WebSocket) ou "ws". O servidor é especificado pelo número da porta do servidor web e nome do host da instância que define a classe Chat.Server (essas informações estão contidas na variável window.location.host). O nome de nossa classe no servidor (Chat.Server.cls) está incluído no URI de abertura do WebSocket como uma solicitação GET para o recurso no servidor.

O evento ws.onopen é disparado quando a conexão WebSocket é estabelecida com êxito, fazendo a transição de um estado de ***conectando*** para um estado ***aberto***.

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

```
    };
```

Este evento atualiza o cabeçalho da janela do chat para indicar que o cliente e o servidor estão conectados.

Enviando mensagens

A ação de um usuário enviando uma mensagem aciona a função de envio. Essa função atua como um invólucro em torno do método ws.send, que contém a mecânica de envio da mensagem do cliente ao servidor pela conexão 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 envia pacotes as informações a serem enviadas ao servidor em um objeto JSON, definindo pares de chave/valor de acordo com o tipo de informação que está sendo enviada (atualização de alias ou mensagem geral). btoa traduz o conteúdo de uma mensagem geral em uma string ASCII codificada em base 64.

Recebendo mensagens

Quando o cliente recebe uma mensagem do servidor, o evento ws.onmessage é acionado.

```
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;
    }
};
```

Dependendo do tipo de mensagem que o cliente recebe (“Chat”, “userlist” ou “status”), o evento onmessage chama wrapmessage ou wrapuser para popular as seções apropriadas da janela de chat com os dados de entrada. Se a mensagem recebida for uma atualização de status, o cabeçalho de status da janela do chat é atualizado com o ID do WebSocket, que identifica a conexão WebSocket bidirecional associada à janela do chat.

Componentes adicionais do cliente

Um erro na comunicação entre o cliente e o servidor aciona o método WebSocket onerror, que emite um alerta que nos notifica do erro e atualiza o cabeçalho de status da página.

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

O método onclose é disparado quando a conexão WebSocket entre o cliente e o servidor é fechada, e atualiza o cabeçalho de status.

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

O servidor

O lado do servidor da aplicação de chat é implementado pela classe Chat.Server, que estende de %CSP.WebSocket. Nossa classe de servidor herda várias propriedades e métodos de %CSP.WebSocket, alguns dos quais discutirei abaixo. Chat.Server também implementa métodos personalizados para processar mensagens de entrada e transmitir mensagens para o(s) cliente(s).

Antes de iniciar o servidor

OnPreServer() é executado antes de o servidor WebSocket ser criado e é herdado da classe %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 define o parâmetro da classe SharedConnection em 1, indicando que nossa conexão WebSocket será assíncrona e suportada por vários processos que definem conexões entre a instância InterSystems IRIS e o Web Gateway. O parâmetro SharedConnection só pode ser alterado em OnPreServer(). O OnPreServer() também armazena o ID WebSocket associado ao cliente no ^Chat.WebSocketConnections global.

O método do servidor

O corpo principal da lógica executada pelo servidor está contido no método Server().

```
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 lê as mensagens recebidas do cliente (usando o método Read da classe %CSP.WebSockets), adiciona os objetos JSON recebidos ao ^Chat.Messages global e chama o ProcessMessage() para encaminhar a mensagem a todos os outros clientes de chat conectados. Quando um usuário fecha sua janela de chat (encerrando assim a conexão WebSocket com o servidor), a chamada do método Server() para Read retorna um código de erro que avalia a macro $$$CSPWebSocketClosed e o método prossegue para tratar o encerramento de acordo.

Processamento e distribui ç ão de mensagens

ProcessMessage() adiciona metadados à mensagem de chat recebida e chama o SendData(), passando a mensagem como um parâmetro.

```
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 a mensagem formatada em JSON do ^Chat.Messages global e a converte em um

objeto do InterSystems IRIS usando a classe %DynamicObject e o método '%FromJSON. Isso nos permite editar facilmente os dados antes de encaminharmos a mensagem para todos os clientes de chat conectados. Adicionamos um atributo Type com o valor "Chat", que o cliente usa para determinar como lidar com a mensagem recebida. SendData() envia a mensagem para todos os outros 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() converte o objeto do InterSystems IRIS de volta em uma string JSON (data.%ToJSON()) e envia a mensagem para todos os clientes de chat. SendData() obtém o ID WebSocket associado a cada conexão cliente-servidor do ^Chat.WebSocketConnections global e usa o ID para abrir uma conexão WebSocket por meio do método OpenServer da classe %CSP.WebSocket. Podemos usar o método OpenServer para fazer isso porque nossas conexões WebSocket são assíncronas - extraímos do pool existente de processos do IRIS-Web Gateway e atribuímos um ID WebSocket que identifica a conexão do servidor a um cliente de chat específico. Por fim, o método Write() %CSP.WebSocket envia a representa ç ão da string JSON da mensagem para o cliente.

Conclusão

Esta aplica ç ão de chat demonstra como estabelecer conexões WebSocket entre um cliente e um servidor hospedado pelo InterSystems IRIS. Para continuar lendo sobre o protocolo e sua implementa ç ão no InterSystems IRIS, dê uma olhada nos links na introdu ç ão.

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

URL de origem: https://pt.community.intersystems.com/post/um-tutorial-sobre-websockets