記事
Toshihiko Minamoto · 2021年4月5日 11m read

WebSocketのチュートリアル

はじめに

Webで行われるサーバーとクライアント間のほとんどの通信は、リクエストとレスポンスの構造に基づいており、 クライアントがサーバーにリクエストを送信すると、サーバーがそのリクエストに対するレスポンスを送信します。 WebSocketプロトコルは、サーバーとクライアント間の双方向通信チャンネルを提供するプロトコルで、サーバーがリクエストを受信しなくても、クライアントにメッセージを送信することができます。
WebSocketプロトコルと、InterSystems IRISでの実装についての詳細は、以下のリンクをご覧ください。

- WebSocketプロトコル
- InterSystems IRISでのWebSocketsに関するドキュメント

このチュートリアルは、「非同期WebSocket -- クイックチュートリアル」を、Caché 2016.2以上とInterSystems IRIS 2018.1以上向けに更新したものです。

非同期動作と同期動作

InterSystems IRISでは、WebSocket接続を同期的または非同期的に実装することができます。 クライアントとサーバー間のWebSocket接続がどのように動作するかは、%CSP.WebSocketクラスの「SharedConnection」プロパティによって決まります。

- SharedConnection=1: 非同期動作
- SharedConnection=0: 同期動作

クライアントとInterSystems IRISインスタンスがホスティングされているサーバーとの間のWebSocket接続には、IRISインスタンスとWebゲートウェイとの間のコネクションが含まれます。
WebSocketの同期動作では、そのコネクションはプライベートチャンネルが使用されますが、 非同期動作では、複数のWebSocketクライアントで、IRISインスタンスとWebゲートウェイとの間のコネクションが共有されます。 WebSocketの非同期動作は、各クライアントを処理するWebゲートウェイとIRISインスタンス間のコネクションを排他的に行う必要がないため、同一のサーバーに多数のクライアントが接続する場合に、そのメリットを発揮します。

このチュートリアルでは、WebSocketの非同期動作を行います。 したがって、開いているすべてのチャットウィンドウは、Webゲートウェイと、WebSocketサーバークラスをホストするIRISインスタンス間のコネクションを共有することになります。

チャットアプリケーションの概要

WebSocketの「hello world」は、ユーザーがそのアプリケーションにログインしているすべてのユーザーにブロー

ドキャストされるメッセージを送信できるチャットアプリケーションです。
このチュートリアルでは、チャットアプリケーションには次のコンポーネントが含まれます。

- サーバー: %CSP.WebSocketを継承したクラスに実装されています。
- クライアント: CSPページで実装されています。

このチャットアプリケーションの実装では、次の内容を実現します。

- ユーザーは、開いているすべてのチャットウィンドウにメッセージをブロードキャストできます。
- オンラインユーザーは、開いているすべてのチャットウィンドウの「オンラインユーザー」リストに表示されます。
- ユーザーは、「alias」キーワードで始まるメッセージを作成することで、ユーザー名を変更できます。このメッセージはブロードキャストされませんが、「オンラインユーザー」リストの内容を更新します。
- ユーザーがチャットウィンドウを閉じると、「オンラインユーザー」リストから削除されます。

チャットアプリケーションのソースコードについては、こちらのGitHubリポジトリをご覧ください。

クライアント

チャットアプリケーションのクライアント側は、チャットウィンドウのスタイル定義、WebSocket接続の宣言、サーバー間との通信を処理するWebSocketのイベントとメソッド、およびサーバーに送信されるメッセージをパッケージ化し、受信メッセージを処理するヘルパー関数を含むCSPページによって実装されます。

まず、アプリケーションが、JavaScript
WebSocketライブラリを使用して、WebSocket接続をどのように初期化するのか確認しましょう。

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

new は、WebSocketクラスの新しいインスタンスを作成します。 これにより、「wss」（WebSocket通信チャンネルにTLSを使用することを示します）または「ws」プロトコルを使って、サーバーにWebSocket接続が開きます。 サーバーは、Chat.Server
クラスを定義するウェブサーバーのポート番号とインスタンスのホスト名で指定されてます（この情報は window.location.host 変数に格納されます）。 サーバークラスの名前（Chat.Server.cls）は、サーバー上のリソースのGETリクエストとして、WebSocketの開始URIに含まれます。

WebSocket接続の確立に成功すると ws.onopen イベントが発生し、***接続中***から***接続***に状態が移行します。

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

このイベントにより、チャットウィンドウの見出しが変更され、クライアントとサーバーが接続されていることが示されるようになります。

メッセージの送信

ユーザーがメッセージを送信するアクションによって、send 関数がトリガーされます。 この関数は ws.send メソッドを囲むラッパーとして機能し、クライアントのメッセージをWebSocket接続を介してサーバーに送信する仕組みが含まれます。

```
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 は、サーバーに送信される情報（エイリアスの更新または一般的なメッセージ）をJSONオブジェクトにパッケージ化し、送信される情報の種類に応じたキー/値ペアを定義します。 btoa
は、一般メッセージの内容を、base-64でエンコードされたASCII文字列に変換します。

メッセージの受信

クライアントがサーバーからメッセージを受信すると、ws.onmessage イベントがトリガーされます。

```
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;
    }
};
```

クライアントが受信するメッセージの種類（「チャット」、「ユーザーリスト」、「ステータス」）に応じて、onmessage イベントによって、wrapmessage または wrapuser
が呼び出され、チャットウィンドウの該当するセクションに受信データを取り込みます。
受信メッセージがステータス更新であれば、チャットウィンドウのステータスの見出しがWebSocket IDで更新されます。このIDは、チャットウィンドウに関連付けられた双方向WebSocket接続を識別するものです。

その他のクライアントコンポーネント

クライアントとサーバー間の通信でエラーが発生すると、WebSocketの onerror メソッドがトリガーされ、エラーの発生を通知するアラートの発行とページのステータス見出しの更新が行われます。

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

クライアントとサーバー間のWebSocket接続が閉じると、onclose
メソッドがトリガーされ、ステータスの見出しが更新されます。

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

サーバー

チャットアプリケーションのサーバー側は、%CSP.WebSocket を拡張した Chat.Server クラスで実装されます。サーバークラスは、%CSP.WebSocketのプロパティとメソッドを継承しています。これについては以下の方で説明します。 Chat.Server は、クライアントからのメッセージを処理するカスタムメソッドと、クライアントにメッセージをブロードキャストするカスタムメソッドも実装します。

サーバーの起動前処理

OnPreServer() は、WebSocketサーバーが作成されて %CSP.WebSocket
クラスから継承される前に実行されます。

```
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
}
```

このメソッドは、SharedConnection
クラスのパラメーターを1に設定し、WebSocket接続が非同期であり、InterSystems

IRISインスタンスとWebゲートウェイ間の接続を定義する複数のプロセスでサポートされることを示します。
SharedConnection パラメーターは、OnPreServer() でしか変更できません。 また、OnPreServer()
は、クライアントに関連付けられているWebSocket IDを ^Chat.WebSocketConnections
グローバルに格納します。

Serverメソッド

サーバーが実行するロジックの本文は、Server() メソッドに含まれます。

```
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
}
```

このメソッドは、クライアントからの受信メッセージを読み取り（%CSP.WebSockets クラスの Read
メソッド）、取得したJSONオブジェクトを ^Chat.Messages
グローバルに追加し、接続されているその他すべてのチャットクライアントにメッセージを転送するための
ProcessMessage() を呼び出します。
ユーザーがチャットウィンドウを閉じると（つまり、サーバーへのWebSocket接続が終了すると）、Server()
メソッドの Read の呼び出しによって、それが評価するエラーコードが $$$CSPWebSocketClosed
マクロに返され、メソッドはそれに応じて閉鎖の処理に進みます。

メッセージの処理と配信

ProcessMessage() は、受信チャットメッセージにメタデータを追加して SendData()
を呼び出し、メッセージをパラメーターとして渡します。

```
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() は、^Chat.Messages グローバルからJSONの書式付きメッセージを取得し、%DynamicObject クラスの %FromJSON メソッドを使って、そのメッセージをInterSystems IRIS オブジェクトに変換します。 この変換により、メッセージが接続されているすべてのチャットクライアントに転送される前に、データを編集しやすくなります。 Type
属性を値「Chat」で追加し、クライアントはこれを使用して、受信メッセージの処理方法を決定しアンス。
SendData() は、接続されているその他すべてのチャットクライアントにメッセージを送信します。

```
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() は、InterSystems IRISオブジェクトをJSON文字列に変換し直し（data.%ToJSON()）、すべてのチャットクライアントにメッセージをプッシュします。 SendData()
は、クライアントとサーバー間のそれぞれの接続に関連付けられたWebSocket IDを
^Chat.WebSocketConnections グローバルから取得し、そのIDを使って、%CSP.WebSocket クラスの
OpenServer メソッドで、WebSocket接続を開きます。 OpenServer メソッドを使ってこの処理を実行できるのは、WebSocket接続が非同期であるからです。IRIS-Webゲートウェイの既存のプールからプロセスをプルし、特定のチャットクライアントへのサーバー接続を識別するWebSocket IDを割り当てています。 最後に、 Write()
%CSP.WebSocket メソッドによって、JSON
文字列に変換されたメッセージがクライアントにプッシュされます。

まとめ

このチャットアプリケーションでは、クライアントとInterSystems
IRISをホストするサーバー間にWebScocket接続を確立する方法が示されています。 InterSystems IRISにおけるプロコルとその実装については、引き続き、「はじめに」セクションに記載されているリンクをご覧ください。

#JavaScript #Node.js #ObjectScript #チュートリアル #フロントエンド #ベストプラクティス #Caché #InterSystems IRIS #Open Exchange

---

ソースURL:
https://jp.community.intersystems.com/post/websocket%E3%81%AE%E3%83%81%E3%83%A5%E3%83%BC%E3%83%88%E3%83%AA%E3%82%A2%E3%83%AB