
# WebSocket の使用 (RFC 6455)

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

WebSocket の使用 (RFC 6455)
InterSystems Version 2024.1 2024-06-03


インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

# 目次

# WebSocket の使用 (RFC 6455)

WebSocket プロトコル (RFC 6455) は、クライアントとサーバの間で全二重のメッセージ指向の通信チャンネルを指定することによって、サーバがクライアントにメッセージを事前にプッシュできるようにするという、基本的な要件に対処します。このプロトコルは、クライアントとサーバの間に既に確立されている標準 TCP チャンネルを介して動作するように (したがって保護されるように) 設計されており、Web ブラウザと Web サーバの間で HTTP プロトコルをサポートするために使用されています。

WebSocket プロトコルおよびその API は、W3C によって標準化され、クライアント部分は HTML 5 に組み込まれています。

媒体 (プロキシやファイアウォールなど) は、WebSocket プロトコルを認識していること (およびサポートしていること) が期待されています。

## 1 WebSocket プロトコル

WebSocket の作成には、クライアントとサーバの間のメッセージの順序付けされた交換が含まれます。まず、WebSocket ハンドシェイクを実行する必要があります。ハンドシェイクは、HTTP メッセージ交換に基づいていて、これに似ています。そのため、既存の HTTP インフラストラクチャ使用して問題なく渡すことができます。

- クライアントは、WebSocket 接続のハンドシェイク要求を送信します。
- サーバは、ハンドシェイク応答を送信します (可能な場合)。

Web サーバは、ハンドシェイク要求メッセージの従来の HTTP ヘッダ構造を認識し、WebSocket プロトコルをサポートしていることを示す、同じように構成された応答メッセージをクライアントに送信します (応答メッセージを送信できることが前提)。両者が同意した場合、チャンネルは HTTP (http://) から WebSocket プロトコル (ws://) に切り替わります。

- プロトコルが正常に切り替わると、チャンネルは、クライアントとサーバの間の全二重通信を許可します。
- 個別のメッセージのデータ・フレーミングは最小です。

クライアントからの典型的な WebSocket ハンドシェイク・メッセージ

```
GET /csp/user/MyApp.MyWebSocketServer.cls HTTP/1.1
Host: localhost
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://localhost
```

サーバからの典型的な WebSocket ハンドシェイク・メッセージ

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

プロトコルを HTTP から WebSocket にアップグレードすることをクライアントのハンドシェイク・メッセージが要求する方法に注意してください。クライアント (Sec-WebSocket-Key) とサーバ (Sec-WebSocket-Accept) の間の一意のキーの交換にも注意してください。

# 2 WebSocket のクライアント・コード (JavaScript)

ブラウザ環境では、クライアントは JavaScript を使用します。標準のテキスト本に使用法のモデルについて詳細に説明されています。ここでは、その基本について簡単に説明します。

## 2.1 WebSocket の作成

最初のパラメータは、WebSocket アプリケーションのサーバ側を特定する URL を表します。2 番目のパラメータは任意です。これがある場合、WebSocket 接続が成功することをサーバがサポートする必要がある、サブプロトコルを指定します。

```
var ws = new WebSocket(url, [protocol]);
```

例:

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

基盤となるトランスポートが SSL/TLS を使用して保護されているかどうかに応じて、プロトコルが `ws` または `wss` に定義される方法に注意してください。

読み取り専用属性 `ws.readyState` は、接続の状態を定義します。以下の値のいずれかになります。

- 0：接続はまだ確立されていません。
- 1：接続が確立され、通信可能です。
- 2：接続は閉じているハンドシェイクに従います。
- 3：接続が閉じられているかまたは開くことができませんでした。

読み取り専用属性 `ws.bufferedAmount` は、send() メソッドを使用してキューに入っている UTF-8 テキストのバイト数を定義します。

## 2.2 WebSocket のクライアント・イベント

以下のイベントを使用できます。

- `ws.onopen`：ソケット接続が確立されるとトリガされます。
- `ws.onmessage`：クライアントがサーバからデータを受信するとトリガされます。

event.data で受信されるデータ。

- `ws.onerror`：通信でエラーが発生するとトリガされます。
- `ws.onclose`：接続が閉じられるとトリガされます。

## 2.3 WebSocket のクライアント・メソッド

以下のメソッドを使用できます。

- `ws.send(data)`：クライアントにデータを送信します。
- `ws.close()`：接続を閉じます。

# 3 WebSocket のサーバ・コード

WebSocket サーバを実装するベース InterSystems IRIS® クラスは %CSP.WebSocket です。

クライアントが WebSocket 接続を要求すると、初期 HTTP 要求 (初期ハンドシェイク・メッセージ) は、アプリケーションの WebSocket サーバを初期化するように CSP エンジンに指示します。WebSocket サーバは、要求元の URL で名前を付けられたクラスです。例えば、WebSocket サーバが MyApp.MyWebSocketServer という名前で、USER ネームスペースで動作するように設計されている場合、WebSocket 接続を要求するために使用される URL は以下のようになります。

```
/csp/user/MyApp.MyWebSocketServer.cls
```

## 3.1 WebSocket のサーバ・イベント

WebSocket サーバを実装するには、%CSP.WebSocket のサブクラスを作成し、そのクラスに必要に応じてコールバックを定義します。これらのメソッドのいずれかを呼び出す前に、Web セッションがロック解除されることに注意してください。

OnPreServer()

このメソッドを実装して、WebSocket サーバが確立される前に実行する必要があるコードを呼び出します。SharedConnection プロパティの変更はここで行う必要があります。

Server() (required)

このメソッドを実装して、WebSocket サーバを作成します。これは、WebSocket アプリケーションのサーバ側の実装です。Read() メソッドおよび Write() メソッドを使用して、クライアントとメッセージを交換できます。EndServer() メソッドを使用して、サーバ側から WebSocket を正常に閉じます。

OnPostServer()

このメソッドを実装して、WebSocket サーバが閉じられた後に実行する必要があるコードを呼び出します。

## 3.2 WebSocket のサーバ・メソッド

これらのコールバック内から以下のメソッドを呼び出すことができます。

Read()

```
Method Read(ByRef len As %Integer = 32656,
     ByRef sc As %Status,
     timeout As %Integer = 86400) As %String
```

このメソッドは、クライアントから len で指定された文字まで読み取ります。呼び出しに成功すると、ステータス (sc) は $$$OK として返されます。失敗すると、以下のエラー・コードのいずれかが返されます。

- $$$CSPWebSocketTimeout : 読み取りメソッドはタイムアウトになりました。
- $$$CSPWebSocketClosed : クライアントが WebSocket を終了しました。

Write()

```
Method Write(data As %String) As %Status
```

このメソッドは、データをクライアントに書き込みます。

### EndServer()

```
Method EndServer() As %Status
```

このメソッドは、クライアントとの接続を閉じることによって WebSocket サーバを正常に終了します。

### OpenServer()

```
Method OpenServer(WebSocketID As %String = "") As %Status
```

このメソッドは、既存の WebSocket サーバを開きます。このメソッドを使用してアクセスできるのは、非同期で動作している WebSocket (`SharedConnection=1`) のみです。

## 3.3 WebSocket のサーバ・プロパティ

これらのコールバック内から以下のプロパティを設定または取得することができます。

### SharedConnection (default: 0)

このプロパティによって、クライアントと WebSocket サーバの間の通信を専用の Web ゲートウェイ接続を介して行う必要があるか、または共有 Web ゲートウェイ接続のプールを介して非同期で行う必要があるかが決まります。このプロパティは、OnPreServer() メソッドで設定する必要があり、以下のように設定できます。

- `SharedConnection=0` : WebSocket サーバは、専用の Web ゲートウェイ接続を介してクライアントと同期して通信します。このモードの処理では、ホスト接続は、アプリケーションの WebSocket サーバに対して事実上「プライベート」です。
- `SharedConnection=1` : WebSocket サーバは、共有 Web ゲートウェイ接続のプールを介してクライアントと非同期で通信します。また、CSP セッション・タイムアウトの期間アクティビティがないと、ソケットがタイムアウトします。

### WebSocketID

このプロパティは、WebSocket の一意の ID を表します。

### SessionId

このプロパティは、WebSocket が作成されたホスト CSP セッション ID を表します。

### BinaryData

このプロパティは、転送されたデータ・ストリームを UTF-8 エンコード・テキストとして解釈する機能をバイパスし、WebSocket フレーム・ヘッダに適切なバイナリ・データ・フィールドを設定するよう Web ゲートウェイに指示します。

これは、バイナリ・データのストリームをクライアントに書き込む前に 1 に設定する必要があります。 次に例を示します。

```
Set ..BinaryData = 1
```

# 4 WebSocket サーバの例

以下の単純な WebSocket サーバ・クラスは、クライアントからの接続を受け入れ、受信したデータを単純にエコー・バックします。

タイムアウトは 10 秒に設定され、Read() メソッドがタイムアウトになるたびに、メッセージがクライアントに書き込まれます。これは、WebSocket を実証する重要な概念の 1 つ (サーバからクライアントとのメッセージ交換の開始) を示します。

最後に、クライアント (つまりユーザ) が文字列 exit を送信すると、WebSocket は正常に閉じます。

```
Method OnPreServer() As %Status
{
   Quit $$$OK
}

Method Server() As %Status
{
   Set timeout=10
   For  {
      Set len=32656
      Set data=..Read(.len, .status, timeout)
      If $$$ISERR(status) {
                           If $$$GETERRORCODE(status) = $$$CSPWebSocketClosed {
                                    Quit
                           }
         If $$$GETERRORCODE(status) = $$$CSPWebSocketTimeout {
               Set status=..Write("Server timed-out at "_$Horolog)
         }
      }
      else {
         If data="exit" Quit
         Set status=..Write(data)
      }
   }
   Set status=..EndServer()
   Quit $$$OK
}

Method OnPostServer() As %Status
{
   Quit $$$OK
}
```

# 5 WebSockets サーバの非同期動作

前の節の例は、専用の InterSystems IRIS 接続を介して、WebSocket サーバがクライアントと同期して動作する例を示しています。このような接続が確立されると、Web ゲートウェイ・システム・ステータス・フォームのステータス列で WebSocket というラベルが付けられます。このモードでは、WebSocket は、ホスト Web セッションのセキュリティ・コンテキスト内で動作し、そのセッションに関連付けられたすべてのプロパティに簡単にアクセスできます。

非同期モードでの動作 (SharedConnection=1) では、ホスト接続は、WebSocket オブジェクトが作成されるとすぐに解放され、クライアントとのその後の対話は、共有接続のプールを介して行われます。クライアントからのメッセージは、Web ゲートウェイ接続の従来のプールを介して InterSystems IRIS に到達し、クライアントへのメッセージは、Web ゲートウェイと InterSystems IRIS の間に確立されたサーバ接続のプールを介して送信されます。

非同期モードでは、WebSocket サーバは、メイン Web セッションから分離されます。つまり、SessionId プロパティは、ホスト・セッション ID の値を保持しますが、セッション・オブジェクトのインスタンスは自動的には作成されません。

前に示した例は、単純に OnPreServer() メソッドの SharedConnection プロパティを設定することによって、非同期で実行できます。ただし、InterSystems IRIS プロセスを永続的に WebSocket と関連付ける必要はありません。Server() メソッドは、WebSocket を閉じずに終了できます (ホスト・プロセスは停止します)。WebSocketID が保持された場合、WebSocket は、次に別の InterSystems IRIS プロセスで開くことができ、クライアントとの通信が再開されます。

以下の例で、MYAPP.SAVE() と MYAPP.RETRIEVE() は WebSocket ID を保存および取得するために作成するカスタム・コードのプレースホルダです。

例：

```
Class MyApp.MyWebSocketServer Extends %CSP.WebSocket
{
    Method OnPreServer() As %Status
    {
        MYAPP.SAVE(..WebSocketID)
        Set ..SharedConnection = 1
        Quit $$$OK
    }

    Method Server() As %Status
    {
        Quit $$$OK
    }

    Method OnPostServer() As %Status
    {
        Quit $$$OK
    }
}
```

WebSocketID は OnPreServer() メソッドを次に使用できるように保持されることに注意してください。また、OnPreServer() メソッドおよび Server() メソッドの SharedConnection プロパティの設定は単に終了することにも注意してください。

WebSocketID の後続の取得：

```
Set WebSocketID = MYAPP.RETRIEVE()
```

クライアントとのリンクの再確立：

```
Set ws=##class(%CSP.WebSocket).%New()
Set %status = ws.OpenServer(WebSocketID)
```

クライアントからの読み取りとクライアントへの書き込み：

```
Set %status=ws.Write(message)
Set data=ws.Read(.len, .%status, timeout)
```

最後に、サーバ側からの WebSocket のクローズ：

```
Set %status=ws.EndServer()
```

# 6 関連項目

- RFC 6455
- クラス・リファレンスの %CSP.WebSocket