Explorando WebSocket

Esta entrada del blog fue organizada con la asistencia de ChatGPT-4o.

Introducción

Hola a todos, soy Li Zhiwei. Como fundador y CTO de la plataforma CodeReview, y ex ingeniero de LeanCloud, tengo una amplia experiencia en WebSocket, especialmente en el desarrollo de SDKs de mensajería instantánea (IM).

La importancia de WebSocket

WebSocket es un protocolo que proporciona un canal de comunicación full-duplex sobre una única conexión TCP. Se utiliza ampliamente en aplicaciones modernas que requieren interacción en tiempo real, como mensajería instantánea, comentarios en tiempo real, juegos multijugador, edición colaborativa y precios de acciones en tiempo real.

Aplicaciones Modernas de WebSocket

WebSocket se utiliza ampliamente en las siguientes áreas: - **Mensajería Instantánea (IM)** - **Comentarios en Tiempo Real** - **Juegos Multijugador** - **Edición Colaborativa** - **Precios de Acciones en Tiempo Real**

La evolución de WebSocket

Sondeo (Polling): El cliente solicita frecuentemente actualizaciones al servidor. **Sondeo largo (Long Polling):** El servidor mantiene la solicitud abierta hasta que hay nueva información disponible. **Conexión bidireccional HTTP:** Requiere múltiples conexiones para enviar y recibir, y cada solicitud incluye cabeceras HTTP. **Conexión única TCP (WebSocket):** Supera las limitaciones de la conexión bidireccional HTTP, ofreciendo mayor capacidad en tiempo real y menor latencia.

Implementación de WebSocket en iOS

WebSocket es un protocolo de comunicación que permite una conexión bidireccional entre un cliente y un servidor. En iOS, puedes utilizar la clase `URLSessionWebSocketTask` para implementar WebSocket en tu aplicación. A continuación, te guiaré a través de los pasos básicos para configurar y utilizar WebSocket en una aplicación iOS.

1. Crear una instancia de `URLSessionWebSocketTask` Primero, necesitas crear una instancia de `URLSessionWebSocketTask` utilizando una URL de WebSocket. Aquí tienes un ejemplo de cómo hacerlo:

```
import Foundation

let url = URL(string: "wss://your.websocket.server")!
let webSocketTask = URLSession.shared.webSocketTask(with: url)
```

2. Conectar al servidor WebSocket Una vez que hayas creado la instancia de `URLSessionWebSocketTask`, puedes conectarte al servidor WebSocket utilizando el método `resume()`:

```
webSocketTask.resume()
```

3. Enviar mensajes al servidor Para enviar un mensaje al servidor, puedes utilizar el método `send(_:)` de `URLSessionWebSocketTask`. Aquí tienes un ejemplo de cómo enviar un mensaje de texto:

```
let message = URLSessionWebSocketTask.Message.string("Hello, WebSocket!")
webSocketTask.send(message) { error in
    if let error = error {
        print("Error sending message: \(error)")
    }
}
```

4. Recibir mensajes del servidor Para recibir mensajes del servidor, puedes utilizar el método `receive(completionHandler:)`. Este método te permite recibir mensajes de forma asíncrona:

```
webSocketTask.receive { result in
    switch result {
    case .failure(let error):
        print("Error receiving message: \(error)")
    case .success(let message):
        switch message {
        case .string(let text):
            print("Received text: \(text)")
        case .data(let data):
            print("Received data: \(data)")
        @unknown default:
            print("Received unknown message type")
        }
    }
}
```

5. Cerrar la conexión WebSocket Cuando ya no necesites la conexión WebSocket, puedes cerrarla utilizando el método `cancel(with:reason:)`:

```
webSocketTask.cancel(with: .goingAway, reason: nil)
```

6. Manejar errores y reconexiones Es importante manejar los errores y las reconexiones en tu aplicación. Puedes utilizar el método `URLSessionDelegate` para manejar eventos de conexión y desconexión.

```
class WebSocketManager: NSObject, URLSessionWebSocketDelegate {
    var webSocketTask: URLSessionWebSocketTask?

    func connect() {
        let url = URL(string: "wss://your.websocket.server")!
        let session = URLSession(configuration: .default, delegate: self, delegateQueue: OperationQueue
        webSocketTask = session.webSocketTask(with: url)
        webSocketTask?.resume()
    }

    func urlSession(_ session: URLSession, webSocketTask: URLSessionWebSocketTask, didOpenWithProtocol
```

```
        print("WebSocket connected")
    }

    func urlSession(_ session: URLSession, webSocketTask: URLSessionWebSocketTask, didCloseWith closeCo
        print("WebSocket disconnected")
    }
}
```

Conclusión Implementar WebSocket en iOS es relativamente sencillo utilizando las APIs proporcionadas por `URLSession`. Con esta guía, deberías poder configurar una conexión WebSocket, enviar y recibir mensajes, y manejar errores y reconexiones en tu aplicación iOS. ¡Buena suerte con tu implementación!

Bibliotecas populares de WebSocket para iOS: - **SocketRocket (Objective-C, 4910 estrellas)** - **Starscream (Swift, 1714 estrellas)** - **SwiftWebSocket (Swift, 435 estrellas)**

Usando SRWebSocket

1. **Inicialización y conexión:**

   ```
   SRWebSocket *webSocket = [[SRWebSocket alloc] initWithURLRequest:[NSURLRequest requestWithURL:[NSUR
   webSocket.delegate = self;
   [webSocket open];
   ```

2. **Enviar mensaje:**

   ```
   [webSocket send:@"Hello, World!"];
   ```

3. **Recibir mensajes:** Implementa los métodos de `SRWebSocketDelegate` para manejar los mensajes entrantes y eventos.

4. **Manejo de errores y notificaciones de eventos:** Maneja adecuadamente los errores y notifica a los usuarios sobre problemas de conexión.

Explicación detallada del protocolo WebSocket

WebSocket opera sobre TCP e introduce varias mejoras: - **Modelo de seguridad:** Añade un modelo de verificación de seguridad basado en el origen del navegador. - **Direccionamiento y nomenclatura de protocolos:** Soporta múltiples servicios en un solo puerto y múltiples

nombres de dominio en una sola dirección IP. - **Mecanismo de tramas:** Mejora TCP con un mecanismo de tramas similar a los paquetes IP, sin límite de longitud. - **Protocolo de cierre:** Asegura un cierre limpio de la conexión.

El núcleo del protocolo WebSocket

El protocolo WebSocket es un protocolo de comunicación bidireccional que permite la transmisión de datos en tiempo real entre un cliente y un servidor. A diferencia de HTTP, que es un protocolo de solicitud-respuesta, WebSocket mantiene una conexión persistente entre el cliente y el servidor, lo que permite la transmisión de datos en ambos sentidos sin necesidad de realizar múltiples solicitudes.

Características clave de WebSocket:

1. **Conexión persistente**: Una vez establecida la conexión WebSocket, esta permanece abierta, lo que permite la transmisión de datos en tiempo real sin la sobrecarga de establecer y cerrar conexiones repetidamente.
2. **Bidireccional**: Tanto el cliente como el servidor pueden enviar datos en cualquier momento, lo que facilita la comunicación en tiempo real.
3. **Baja latencia**: Debido a la naturaleza persistente de la conexión, la latencia en la transmisión de datos es significativamente menor en comparación con HTTP.
4. **Soporte para datos binarios y de texto**: WebSocket puede transmitir tanto datos binarios como de texto, lo que lo hace versátil para una variedad de aplicaciones.

Establecimiento de la conexión WebSocket El proceso de establecimiento de una conexión WebSocket comienza con un "handshake" HTTP. El cliente envía una solicitud HTTP al servidor con un encabezado especial `Upgrade: websocket`, indicando que desea cambiar al protocolo WebSocket. Si el servidor acepta la solicitud, responde con un código de estado `101 Switching Protocols`, y la conexión se actualiza a WebSocket.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
```

```
Sec-WebSocket-Version: 13
```

El servidor responde con:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Una vez completado el handshake, la conexión se convierte en una conexión WebSocket y puede comenzar la transmisión de datos.

Transmisión de datos Los datos en WebSocket se transmiten en forma de "frames". Un frame puede contener datos de texto o binarios, y puede ser enviado en cualquier dirección. Los frames están diseñados para ser eficientes y tienen un encabezado pequeño que incluye información como el tipo de frame y la longitud de los datos.

```
// Ejemplo de envío de datos desde el cliente
const socket = new WebSocket('ws://example.com/chat');

socket.onopen = function(event) {
    socket.send('Hola, servidor!');
};

socket.onmessage = function(event) {
    console.log('Mensaje recibido del servidor: ' + event.data);
};
```

Cierre de la conexión La conexión WebSocket puede cerrarse en cualquier momento por cualquiera de las partes (cliente o servidor). El cierre se realiza enviando un frame de cierre, que puede incluir un código de estado y un mensaje opcional que indica la razón del cierre.

```
// Ejemplo de cierre de conexión desde el cliente
socket.close(1000, 'Cierre normal');
```

Conclusión El protocolo WebSocket es una herramienta poderosa para aplicaciones que requieren comunicación en tiempo real, como chats en línea, juegos multijugador y actualizaciones en vivo. Su capacidad para mantener una conexión persistente y bidireccional lo hace ideal para escenarios donde la latencia y la eficiencia son críticas.

1. Apretón de manos (Handshake): El apretón de manos de WebSocket utiliza el mecanismo de actualización de HTTP: - **Solicitud del cliente:** `http GET /chat HTTP/1.1 Host: server.example.com Upgrade: websocket Connection: Upgrade Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== Origin: http://example.com Sec-WebSocket-Protocol chat, superchat Sec-WebSocket-Version: 13`

- **Respuesta del servidor:** `http HTTP/1.1 101 Switching Protocols Upgrade: websocket Connection: Upgrade Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo= Sec-WebSocket-Protocol: chat`

2. Transferencia de datos: Los frames de WebSocket pueden contener texto en UTF-8, datos binarios y frames de control, como cierre, ping y pong.

3. Seguridad: El navegador agrega automáticamente la cabecera `Origin`, la cual no puede ser falsificada por otros clientes.

URI de WebSocket

- **ws-URI** `ws://host:port/path?query`
- **wss-URI** `wss://host:port/path?query`

Protocolo de tramas WebSocket

Estructura del Frame: - **FIN (1 bit):** Indica si este es el último fragmento del mensaje. - **RSV1, RSV2, RSV3 (1 bit cada uno):** Reservados para uso futuro. - **Opcode (4 bits):** Define cómo se debe interpretar la carga útil. - `0x0`: Frame de continuación - `0x1`: Frame de texto - `0x2`: Frame binario - `0x8`: Cierre de conexión - `0x9`: Ping - `0xA`: Pong - **Mask (1 bit):** Indica si la carga útil está enmascarada. - **Longitud de la carga útil (7 bits):** Longitud de los datos de la carga útil.

Clave de máscara: Se utiliza para prevenir ataques de intermediario al enmascarar los fotogramas del cliente.

Cierre del apretón de manos (Handshake)

Marco de cierre: - Puede contener un cuerpo que indique la razón del cierre. - Ambas partes deben enviar y responder al marco de cierre.

Ejemplo

Ejemplo 1: Mensaje de texto sin enmascarar en un solo frame

```
0x81 0x05 0x48 0x65 0x6c 0x6c 0x6f
```

Contiene “Hello”

Ejemplo 2: Mensaje de texto enmascarado en un solo marco

```
0x81 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58
```

Contiene “Hello”, con clave de enmascaramiento.

Ejemplo 3: Mensaje de texto fragmentado sin enmascarar

```
0x01 0x03 0x48 0x65 0x6c
0x80 0x02 0x6c 0x6f
```

El mensaje fragmentado contiene dos tramas: “Hel” y “lo”.

Temas avanzados

Enmascaramiento y Desenmascaramiento: - El enmascaramiento se utiliza para prevenir ataques de intermediario (man-in-the-middle). - Cada trama enviada desde el cliente debe estar enmascarada. - La clave de enmascaramiento para cada trama se selecciona aleatoriamente.

Fragmentación: - Se utiliza para enviar datos de longitud desconocida. - Los mensajes fragmentados comienzan con un frame donde FIN es 0 y terminan con un frame donde FIN es 1.

Marcos de Control: - Los marcos de control (como cierre, ping y pong) tienen códigos de operación específicos. - Estos marcos se utilizan para gestionar el estado de la conexión WebSocket.

Escalabilidad

Los datos de extensión pueden colocarse antes de los datos de la aplicación en el cuerpo del mensaje: - Los bits reservados pueden controlar cada trama. - Se reservan algunos códigos de operación para futuras definiciones. - Si se necesitan más códigos de operación, se pueden utilizar los bits reservados.

Envío: - Es necesario asegurarse de que la conexión esté en estado OPEN. - Los datos se encapsulan en tramas, y si los datos son demasiado grandes, se puede optar por enviarlos en fragmentos. - El valor de la primera trama debe ser correcto, indicando al receptor el tipo de datos (texto o binario). - El FIN de la última trama debe estar configurado en 1.

Cierre del apretón de manos (Handshake): - Ambas partes pueden enviar un marco de cierre. - Después de enviar el marco de cierre, no se envía más datos. - Al recibir un marco de cierre, se descartan todos los datos recibidos posteriormente.

Cerrar la conexión: - Cerrar la conexión WebSocket, es decir, cerrar la conexión TCP subyacente. - Después de enviar o recibir un marco de cierre, el estado de la conexión WebSocket es “cerrando”. - Cuando la conexión TCP subyacente se cierra, el estado de la conexión WebSocket es “cerrada”.

Referencias

- WebSocket RFC: RFC6455
- Zhihu¿Cuál es el principio de WebSocket?: Enlace de Zhihu
- SocketRocket: Enlace de GitHub

Agradecimientos

Gracias a todos por su atención. Si tienen más preguntas o desean discutir algún tema, no duden en contactarme en GitHub o Weibo.