

前端处理连续返回消息 (Front-end handles continuous response)



Exported by ChatGPT Exporter at 2023-05-10T09:11:34.460Z

修改这段代码，允许后端连续返回结果

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ChatSydney</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
<div class="container">
  <div class="chat-history">
    <h3>Chat History:</h3>
    <div class="messages" id="messages"> </div>
  </div>
  <div class="user-input">
    <h3>User Input:</h3>
    <textarea id="userInput" rows="5"> </textarea>
    <button id="sendBtn">Send</button>
    <select id="send-mode-selector" onchange="enterMode = this.value">
      <option value="enter">Press Enter to send</option>
      <option value="ctrl-enter">Press Ctrl+Enter to send</option>
    </select>
  </div>
</div>
<script>
  const messages = document.getElementById('messages');
  const userInput = document.getElementById('userInput');
  const sendBtn = document.getElementById('sendBtn');
  let enterMode = "enter";

  userInput.addEventListener('keydown', async (event) => {
    if (enterMode === "enter" && event.key === 'Enter' && !event.ctrlKey) {
```

```
        event.preventDefault();
        await sendMessage();
    } else if (enterMode === "ctrl-enter" && event.key === 'Enter' && event.ctrlKey) {
        event.preventDefault();
        await sendMessage();
    }
});
```

```
async function sendMessage() {
    const inputText = userInput.value;
    if (inputText.trim() === '') return;

    appendMessage('user', inputText);
    userInput.value = '';

    try {
        const response = await sendMessageToBackend(inputText);
        appendMessage('assistant', JSON.stringify(response));
    } catch (error) {
        alert(error);
    }
}
```

```
sendBtn.addEventListener('click', sendMessage);
```

```
function appendMessage(sender, text) {
    const messageElement = document.createElement('div');
    messageElement.classList.add(sender === 'user' ? 'user-message' : 'assistant-message');
    messageElement.textContent = text;
    messages.appendChild(messageElement);
    messages.scrollTop = messages.scrollHeight;
}
```

```
let websocket;
```

```
async function connectWebSocket() {
    return new Promise((resolve, reject) => {
```

```
websocket = new WebSocket('ws://localhost:8080');

websocket.onopen = () => {
  resolve();
};

websocket.onerror = (error) => {
  reject(error);
};
});
}

async function sendMessageToBackend(message, context) {
  if (!websocket || websocket.readyState !== WebSocket.OPEN) {
    try {
      await connectWebSocket();
    } catch (error) {
      console.error('WebSocket error:', error);
      return 'Error connecting to the server.';
    }
  }

  return new Promise((resolve, reject) => {
    websocket.send(JSON.stringify({message, context}));

    websocket.onmessage = (event) => {
      const response = JSON.parse(event.data);
      resolve(response.message);
    };

    websocket.onerror = (error) => {
      reject(error);
    };
  });
}
</script>
```

```
</body>
```

```
</html>
```



为了让后端连续返回结果，我们需要修改 `sendMessageToBackend` 函数。在这里，我将修改该函数以处理从后端接收到的多个消息，而不是在每次用户请求时只处理一个消息。

将下面的代码替换为原有的 `sendMessageToBackend` 函数：

```
async function sendMessageToBackend(message, context) {
  if (!websocket || websocket.readyState !== WebSocket.OPEN) {
    try {
      await connectWebSocket();
    } catch (error) {
      console.error('WebSocket error:', error);
      return 'Error connecting to the server.';
    }
  }

  return new Promise((resolve, reject) => {
    websocket.send(JSON.stringify({ message, context }));

    let messageReceived = false;

    websocket.onmessage = (event) => {
      const response = JSON.parse(event.data);
      if (!messageReceived) {
        messageReceived = true;
        resolve(response.message);
      } else {
        appendMessage('assistant', JSON.stringify(response.message));
      }
    };

    websocket.onerror = (error) => {
      reject(error);
    };
  });
}
```

此代码更改将允许您的前端处理连续返回的消息。当收到第一个消息时，Promise 将解析该消息。后续接收到的消息将直接添加到消息窗口中。

请参考以下的代码处理服务端发来的消息：

```
if self.responding:
    return
self.set_responding(True)
user_input = self.user_input.toPlainText()
self.user_input.clear()
self.chat_history.moveCursor(QTextCursor.MoveOperation.End)
text = self.chat_history.toPlainText()
if not text.endswith("\n\n"):
    if text.endswith("\n"):
        self.chat_history.insertPlainText("\n")
    else:
        self.chat_history.insertPlainText("\n\n")
chatbot = await Chatbot.create(cookie_path="cookies.json", proxy="http://127.0.0.1:2081")

async def stream_output():
    self.chat_history.moveCursor(QTextCursor.MoveOperation.End)
    self.chat_history.insertPlainText(f"[user](#message)\n{user_input}\n\n")
    wrote = 0
    async for final, response in chatbot.ask_stream(
        prompt=user_input,
        raw=True,
        webpage_context=self.chat_history.toPlainText(),
        conversation_style="creative",
        search_result=True,
    ):
        if not final and response["type"] == 1 and "messages" in response["arguments"][0]:
            self.chat_history.moveCursor(QTextCursor.MoveOperation.End)
            message = response["arguments"][0]["messages"][0]
            match message.get("messageType"):
                case "InternalSearchQuery":
                    self.chat_history.insertPlainText(
                        f"[assistant](#search_query)\n{message['hiddenText']}\n\n")
```

```

        case "InternalSearchResult":
            self.chat_history.insertPlainText(
                f"[assistant](#search_results)\n{message['hiddenText']}\n\n")
        case None:
            if "cursor" in response["arguments"][0]:
                self.chat_history.insertPlainText(f"[assistant](#message)\n")
                wrote = 0
            if message.get("contentOrigin") == "Apology":
                QErrorMessage(self).showMessage("Message revoke detected")
                break
            else:
                self.chat_history.insertPlainText(message["text"][wrote:])
                wrote = len(message["text"])
                if "suggestedResponses" in message:
                    suggested_responses = list(map(lambda x: x["text"],
message["suggestedResponses"]))
                    self.chat_history.insertPlainText(f"""\n[assistant](#suggestions)
```json
{{"suggestedUserResponses": {suggested_responses}}}
```
\n\n""")
                    break
            if final and not response["item"]["messages"][-1].get("text"):
                raise Exception("Looks like the user message has triggered the Bing filter")

        try:
            await stream_output()
        except Exception as e:
            QErrorMessage(self).showMessage(str(e))
        self.set_responding(False)
        self.chat_history.moveCursor(QTextCursor.MoveOperation.End)
        await chatbot.close()

```



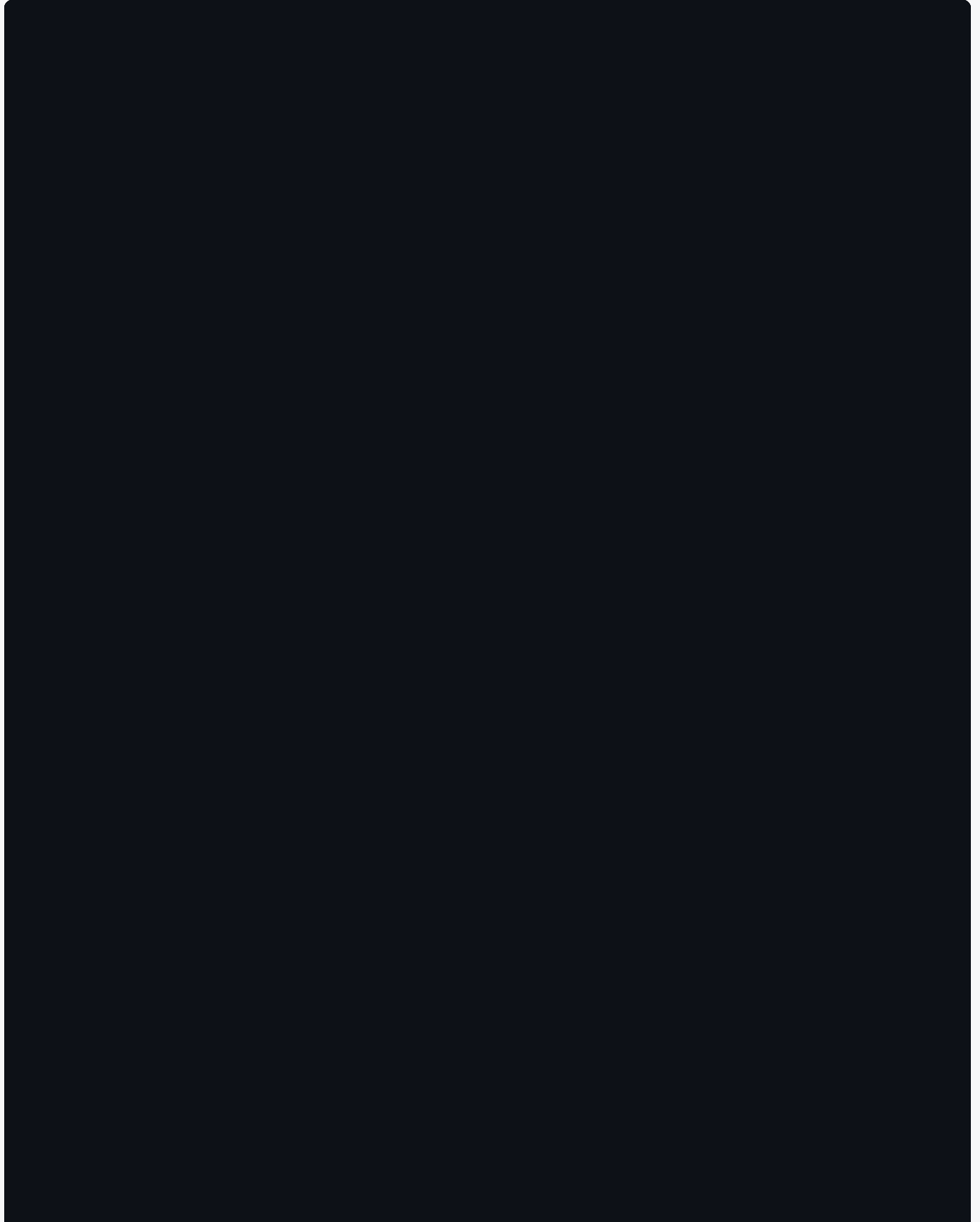
从您的代码中，我看到您正在处理与聊天机器人的流式对话。您已经处理了不同类型的消息，如内部搜索查询、内部搜索结果、普通消息等。

为了更好地帮助您，我需要了解一下您需要解决的问题或改进的方面。请告诉我您在这段代码中遇到的困难或您想要改进的功能。这将帮助我为您提供更有针对性的建议。

请根据我的Qt Python代码改写上面的JS代码



根据您提供的 Qt Python 代码，我修改了上面的 JavaScript 代码以处理不同类型的消息。以下是根据您的需求修改后的 JavaScript 代码：







```
</script>  
</body>  
</html>
```

这里我将原先的 `sendMessageToBackend` 函数

继续



替换为 `streamOutput` 函数，以便更好地匹配您的 Qt Python 代码。`streamOutput` 函数现在处理从服务端接收到的多个消息类型，包括内部搜索查询、内部搜索结果和普通消息。此外，我还添加了处理消息撤回的逻辑。以下是修改后的代码细节：

1. 更改了函数名为 `streamOutput` 以匹配您的 Python 代码。
2. 将输入的用户消息和聊天历史上下文发送到 WebSocket 服务器。
3. 使用 `switch` 语句代替 `if` 语句，根据消息类型处理不同的情况。
4. 添加了撤回消息的处理逻辑，根据 `contentOrigin` 属性判断。

现在，JavaScript 代码应该可以更好地处理从服务端接收到的多个消息类型，类似于您提供的 Qt Python 代码。