

# 利用Tornado 对 WebSocket 支持，实现实时通信应用

Original williamsun2020 攻城狮奶爸杂货铺 2024年08月14日 22:01 安徽

在当今的 Web 应用程序中，**实时通信对于提供无缝和交互式的用户体验至关重要。**  
**WebSocket 已成为实现客户端和服务端之间实时双向通信的热门选择。** Tornado 是一个 Python Web 框架，它为 WebSocket 提供了出色的支持，使其成为构建实时 Web 应用程序的理想选择。

在本博文中，我们将**探讨 Tornado 如何支持 WebSocket**，并通过一个简单的演示来**演示如何使用 Tornado 实现实时通信。**

## 了解 Tornado 中的 WebSocket

WebSocket 是一种通过单个 TCP 连接提供全双工通信通道的协议。与遵循请求-响应模型的 HTTP 不同，WebSocket 允许服务器随时将数据推送到客户端，这使其成为聊天应用程序、实时通知和协作工具等实时应用程序的理想选择。

Tornado 通过其 ***tornado.websocket*** 模块原生支持 WebSocket。此模块提供了一个 ***WebSocketHandler*** 类，您可以扩展它来处理 WebSocket 连接。

## 设置 Tornado

首先，确保 Tornado 已安装在您的 Python 环境中。如果没有，您可以使用 pip 安装它：

```
1 pip install tornado
```

70M

## 在 Tornado 中实现基本 WebSocket 服务器

让我们首先使用 Tornado 创建一个简单的 WebSocket 服务器。该服务器将把客户端发送的消息回显给客户端。

```
1 import tornado.ioloop
2 import tornado.web
3 import tornado.websocket
4
5 class EchoWebSocket(tornado.websocket.WebSocketHandler):
6     def open(self):
7         print("WebSocket opened")
8         self.write_message("Welcome to the Tornado WebSocket server!")
9
10    def on_message(self, message):
11        print(f"Received message: {message}")
12        self.write_message(f"You said: {message}")
13
14    def on_close(self):
15        print("WebSocket closed")
16
17 if __name__ == "__main__":
18     app = tornado.web.Application([
19         (r"/websocket", EchoWebSocket),
20     ])
21     app.listen(8888)
22     print("WebSocket server started at ws://localhost:8888/websocket")
23     tornado.ioloop.IOLoop.current().start()
```

## 解释

1. **WebSocketHandler** : 我们创建一个从 `WebSocketHandler` 继承的类 `EchoWebSocket`。该类处理 `WebSocket` 事件，如打开连接、接收消息和关闭连接。
2. **open()**: 打开新的 `WebSocket` 连接时调用此方法。在这里，您可以初始化资源或向客户端发送初始消息。
3. **on\_message()**: 每当从客户端收到消息时都会触发此方法。在此演示中，我们只是将消息回显给客户端。
4. **on\_close()**: 关闭 `WebSocket` 连接时调用此方法。

## 运行 WebSocket 服务器

运行Python 脚本，`WebSocket` 服务器将开始监听 `ws://localhost:8888/websocket`。

## 创建 WebSocket 客户端

要与我们的 `WebSocket` 服务器交互，让我们创建一个简单的 `HTML` 客户端。

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>WebSocket Client</title>
7 </head>
8 <body>
9     <h1>WebSocket Echo Client</h1>
10    <input type="text" id="messageInput" placeholder="Enter your message">
11    <button onclick="sendMessage()">Send</button>
12    <div id="response"></div>
```

```
13
14     <script>
15         const ws = new WebSocket("ws://localhost:8888/websocket");
16
17         ws.onopen = function() {
18             console.log("Connected to WebSocket server");
19         };
20
21         ws.onmessage = function(event) {
22             document.getElementById("response").innerText = "Server: " + ever
23         };
24
25         ws.onclose = function() {
26             console.log("Disconnected from WebSocket server");
27         };
28
29         function sendMessage() {
30             const message = document.getElementById("messageInput").value;
31             ws.send(message);
32         }
33     </script>
34 </body>
35 </html>
```

## 解释

1. **WebSocket 对象**: 我们通过连接到服务器 URL 来创建 WebSocket 对象。该对象提供发送和接收消息的方法。
2. **onopen**: 当建立 WebSocket 连接时触发此事件。

3. **onmessage**: 当从服务器收到消息时触发此事件。我们在 HTML div 中显示消息。
4. **sendMessage()**: 此函数将用户输入的消息发送到 WebSocket 服务器。

## 运行演示

1. 打开 WebSocket 服务器脚本并运行它。
2. 在浏览器中打开 HTML 文件。
3. 在输入框中输入消息，然后单击“Send”。服务器将向您回显该消息。

## 扩展 WebSocket 服务器

为了演示 Tornado 的实时功能，让我们扩展我们的 WebSocket 服务器以向所有连接的客户端广播消息。

### 修改后的服务器代码

```
1 import tornado.ioloop
2 import tornado.web
3 import tornado.websocket
4
5 clients = []
6
7 class BroadcastWebSocket(tornado.websocket.WebSocketHandler):
8     def open(self):
9         print("WebSocket opened")
10        clients.append(self)
11        self.write_message("Welcome! You are now connected to the WebSocket s
12
```

```
13     def on_message(self, message):
14         print(f"Received message: {message}")
15         for client in clients:
16             if client != self:
17                 client.write_message(message)
18
19     def on_close(self):
20         print("WebSocket closed")
21         clients.remove(self)
22
23 if __name__ == "__main__":
24     app = tornado.web.Application([
25         (r"/websocket", BroadcastWebSocket),
26     ])
27     app.listen(8888)
28     print("WebSocket server started at ws://localhost:8888/websocket")
29     tornado.ioloop.IOLoop.current().start()
```

## 解释

1. 客户端列表：我们维护所有已连接客户端的列表。当打开新的 WebSocket 连接时，客户端将添加到此列表中。
2. 广播：在 on\_message 方法中，我们不会将消息回显到同一个客户端，而是将其广播到除发送方之外的所有已连接客户端。

## 测试广播功能

1. 运行修改后的服务器代码。
2. 在不同的浏览器选项卡或窗口中打开 HTML 客户端的多个实例。
3. 从一个客户端发送一条消息，您将看到它实时出现在所有已连接的客户端中。

## 以下是一些关附加演示

### 1. 实时通知系统

此演示展示了如何实现实时通知系统，当事件发生时，服务器会向所有连接的客户端发送通知。

### 服务器代码

```
1 import tornado.ioloop
2 import tornado.web
3 import tornado.websocket
4 import tornado.gen
5 import time
6 import threading
7
8 clients = []
9
10 class NotificationWebSocket(tornado.websocket.WebSocketHandler):
11     def open(self):
12         clients.append(self)
13         self.write_message("Connected to the notification server.")
14
15     def on_close(self):
16         clients.remove(self)
17
18 def notify_clients(message):
19     for client in clients:
20         client.write_message(message)
```

```
21
22 def notification_emitter():
23     count = 1
24     while True:
25         time.sleep(5)
26         message = f"Notification {count}: Server generated an event!"
27         notify_clients(message)
28         count += 1
29
30 if __name__ == "__main__":
31     app = tornado.web.Application([
32         (r"/websocket", NotificationWebSocket),
33     ])
34     app.listen(8888)
35
36     # Start a background thread to generate notifications
37     notification_thread = threading.Thread(target=notification_emitter)
38     notification_thread.start()
39
40     print("Notification server started at ws://localhost:8888/websocket")
41     tornado.ioloop.IOLoop.current().start()
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>WebSocket Notifications</title>
7 </head>
```



```
8 <body>
9   <h1>Real-time Notifications</h1>
10  <div id="notifications"></div>
11
12  <script>
13      const ws = new WebSocket("ws://localhost:8888/websocket");
14
15      ws.onmessage = function(event) {
16          const notificationDiv = document.createElement("div");
17          notificationDiv.innerText = event.data;
18          document.getElementById("notifications").appendChild(notificationDiv);
19      };
20  </script>
21 </body>
22 </html>
```

## 解释

1. 后台线程：服务器运行一个后台线程，每 5 秒生成一个通知并将其广播给所有连接的客户端。
2. 通知处理：客户端为每个通知动态创建一个新的 div 元素并将其附加到通知列表中。

## 2. 协作绘图应用程序

在此演示中，我们将创建一个协作绘图应用程序，多个用户可以在共享画布上实时绘图。

## 服务器代码

```
1 import tornado.ioloop
2 import tornado.web
3 import tornado.websocket
4
5 clients = []
6
7 class DrawingWebSocket(tornado.websocket.WebSocketHandler):
8     def open(self):
9         clients.append(self)
10
11     def on_message(self, message):
12         for client in clients:
13             if client != self:
14                 client.write_message(message)
15
16     def on_close(self):
17         clients.remove(self)
18
19 if __name__ == "__main__":
20     app = tornado.web.Application([
21         (r"/websocket", DrawingWebSocket),
22     ])
23     app.listen(8888)
24     print("Drawing server started at ws://localhost:8888/websocket")
25     tornado.ioloop.IOLoop.current().start()
```

```
1 <!DOCTYPE html>
2 <html lang="en">
```

```
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Collaborative Drawing</title>
7   <style>
8       canvas {
9           border: 1px solid black;
10      }
11 </style>
12 </head>
13 <body>
14   <h1>Collaborative Drawing</h1>
15   <canvas id="drawingCanvas" width="500" height="500"></canvas>
16
17   <script>
18       const canvas = document.getElementById("drawingCanvas");
19       const ctx = canvas.getContext("2d");
20       const ws = new WebSocket("ws://localhost:8888/websocket");
21
22       let drawing = false;
23
24       canvas.addEventListener("mousedown", function() {
25           drawing = true;
26       });
27
28       canvas.addEventListener("mouseup", function() {
29           drawing = false;
30           ctx.beginPath(); // Start a new path when the mouse is released
31       });
32
```

```
33     canvas.addEventListener("mousemove", function(event) {
34         if (!drawing) return;
35
36         const rect = canvas.getBoundingClientRect();
37         const x = event.clientX - rect.left;
38         const y = event.clientY - rect.top;
39
40         ws.send(JSON.stringify({ x, y }));
41
42         draw(x, y);
43     });
44
45     ws.onmessage = function(event) {
46         const { x, y } = JSON.parse(event.data);
47         draw(x, y);
48     };
49
50     function draw(x, y) {
51         ctx.lineTo(x, y);
52         ctx.stroke();
53         ctx.beginPath();
54         ctx.moveTo(x, y);
55     }
56 </script>
57 </body>
58 </html>
```

## 解释

1. 实时绘图：当用户在其画布，绘图的坐标被发送到 WebSocket 服务器，该服务器将坐标广播给所有连接的客户端。然后客户端实时绘制线条。
2. 画布处理：客户端使用 HTML5 画布和 JavaScript 来处理绘图操作，并在所有连接的客户端之间同步绘图。

### 3. 带用户管理的实时聊天室

此演示通过添加用户管理功能（例如加入和离开通知）扩展了基本聊天应用程序。

#### 服务器代码

```
1 import tornado.ioloop
2 import tornado.web
3 import tornado.websocket
4 import json
5
6 clients = []
7
8 class ChatWebSocket(tornado.websocket.WebSocketHandler):
9     def open(self):
10         clients.append(self)
11         self.write_message(json.dumps({"type": "info", "message": "Welcome to"}))
12         notify_users("A new user has joined the chat.")
13
14     def on_message(self, message):
15         notify_users(message, exclude_self=False)
16
17     def on_close(self):
18         clients.remove(self)
19         notify_users("A user has left the chat.")
```

```
20
21 def notify_users(message, exclude_self=True):
22     for client in clients:
23         if exclude_self and client == tornado.websocket.WebSocketHandler:
24             continue
25         client.write_message(json.dumps({"type": "chat", "message": message}))
26
27 if __name__ == "__main__":
28     app = tornado.web.Application([
29         (r"/websocket", ChatWebSocket),
30     ])
31     app.listen(8888)
32     print("Chat server started at ws://localhost:8888/websocket")
33     tornado.ioloop.IOLoop.current().start()
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Real-time Chat Room</title>
7 </head>
8 <body>
9     <h1>Chat Room</h1>
10    <div id="chatMessages"></div>
11    <input type="text" id="chatInput" placeholder="Type a message">
12    <button onclick="sendMessage()">Send</button>
13
```

```
14     <script>
15         const ws = new WebSocket("ws://localhost:8888/websocket");
16
17         ws.onmessage = function(event) {
18             const { type, message } = JSON.parse(event.data);
19             const messageDiv = document.createElement("div");
20             messageDiv.innerText = type === "info" ? `Info: ${message}` : `Me
21             document.getElementById("chatMessages").appendChild(messageDiv);
22         };
23
24         function sendMessage() {
25             const message = document.getElementById("chatInput").value;
26             ws.send(message);
27             document.getElementById("chatInput").value = '';
28         }
29     </script>
30 </body>
31 </html>
```

## 解释

1. 用户管理：当用户加入或离开聊天室时，会向所有连接的用户广播通知。
2. 消息处理：消息以 JSON 对象的形式发送和接收，允许客户端区分不同类型的消息（例如，信息与聊天）。

## 4. 实时股票价格行情机

在此演示中，我们将模拟一个实时股票价格行情机，它会更新所有连接客户端上的股票价格。

## 服务器代码

```
1 import tornado.ioloop
2 import tornado.web
3 import tornado.websocket
4 import tornado.gen
5 import random
6 import time
7 import threading
8
9 clients = []
10
11 class StockTickerWebSocket(tornado.websocket.WebSocketHandler):
12     def open(self):
13         clients.append(self)
14         self.write_message("Connected to the stock ticker.")
15
16     def on_close(self):
17         clients.remove(self)
18
19 def broadcast_stock_prices():
20     while True:
21         time.sleep(2)
22         stock_price = round(random.uniform(100, 500), 2)
23         for client in clients:
24             client.write_message(f"Stock Price: ${stock_price}")
25
26 if __name__ == "__main__":
27     app = tornado.web.Application([
28         (r"/websocket", StockTickerWebSocket),
```



复制

```
29     ])  
30     app.listen(8888)  
31  
32     # Start a background thread to generate stock prices  
33     stock_thread = threading.Thread(target=broadcast_stock_prices)  
34     stock_thread.start()  
35  
36     print("Stock ticker server started at ws://localhost:8888/websocket")  
37     tornado.ioloop.IOLoop.current().start()
```

```
1  <!DOCTYPE html>  
2  <html lang="en">  
3  <head>  
4      <meta charset="UTF-8">  
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">  
6      <title>Real-time Stock Ticker</title>  
7  </head>  
8  <body>  
9      <h1>Stock Ticker</h1>  
10     <div id="stockPrice"></div>  
11  
12     <script>  
13         const ws = new WebSocket("ws://localhost:8888/websocket");  
14  
15         ws.onmessage = function(event) {  
16             document.getElementById("stockPrice").innerText = event.data;  
17         };  
18     </script>  
19 </body>
```

```
20 </html>
```


解释

- 1. 模拟股价：服务器每2秒随机生成一个股价，并广播给所有连接的客户端。
  - 2. 实时更新：客户端实时显示最新股价从 WebSocket 服务器获取。
- 以上这些演示使用 Tornado 的 WebSocket 支持提供了广泛的实时通信场景，展示了如何构建交互式动态应用程序。

Tornado 中的 WebSockets 提供了一种在 Web 应用程序中实现实时通信的强大方法。通过扩展基本的 WebSocket 服务器，您可以创建复杂的实时系统，如聊天应用程序、实时通知和协作工具。Tornado 框架及其异步功能可确保您的 WebSocket 服务器能够高效处理多个连接。赶紧行动起来吧！有问题留言，让我们共同进步。

往期精彩推荐：

- 1. [深入探究Tornado 的异步 I/O](#)
- 2. [FastAPI 中的高级功能：将 GraphQL 与 FastAPI 集成](#)
- 3. [FastAPI WebSockets：使用 WebSockets 进行实时更新](#)



攻城狮奶爸杂货铺

技术随想，人生感悟，趣闻趣事记录，包罗万象，应有尽有

409篇原创内容

公众号

码字不易，喜欢的话点👍和🌟，分享给更多的朋友。或者关注小编，了解更多新篇章。

也可以通过合集查看更多，谢谢！👉👉👉

# Tornado 16

Tornado · 目录 ≡

< 上一篇

深入探究Tornado 的异步 I/O

下一篇 >

使用 Tornado 创建可扩展的应用程序