天意凉

昵称： 天意凉
园龄： 7年1个月
粉丝： 43
关注： 11
+加关注

< 2024年12月 >

日	一	二	三	四	五	六
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

搜索
找找看

【websocket】小白快速上手flask-socketio

大家好，我是一个初级的Python开发工程师。本文是结合官方教程和代码案例，简单说下我对flask-socketio的使用理解。

一、websocket简介

websocket 说白一点就是，建立客户端和服务端双向通讯通道， 服务器可以主动向客户端发消息。

二、flask-socketio理解与使用

1. 环境准备：Python3.7

```
pip install eventlet==0.33.3
pip install flask-socketio==5.8.0
pip install flask==1.1.4
```

2. 代码来自官方教程

常用链接

我的随笔

我的评论

我的参与

最新评论

我的标签

我的标签

python小灶(17)

python面试常见题(4)

随想(3)

计算机基础知识(2)

网络编程基础(2)

Web前端学习笔记(2)

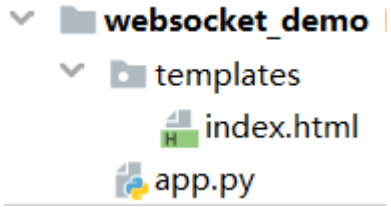
python概念理解区(2)

personal_daily(2)

web开发(1)

下面的代码亲测可用，请放心食用。

（1）项目结构



（2）app.py代码

复制

```
from threading import Lock
from flask import Flask, render_template, session, request, copy_current_request_context
from flask_socketio import SocketIO, emit, join_room, leave_room, close_room, rooms, disconnect

# Set this variable to "threading", "eventlet" or "gevent" to test the
# different async modes, or leave it set to None for the application to choose
# the best option based on installed packages.
async_mode = None

app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret!'
socketio = SocketIO(app, async_mode=async_mode)
thread = None
thread_lock = Lock()


def background_thread():
    """Example of how to send server generated events to clients."""
    count = 0
    while True:
```

websocket(1)
更多

合集

学习心得(3)
flask框架(1)
web开发(6)
web开发面试(6)

随笔分类

personal_daily(3)
scrapy源码解读(2)

随笔档案

2023年7月(2)
2020年4月(2)
2019年1月(8)
2018年1月(13)
2017年12月(1)

```
        socketio.sleep(10)
        count += 1
        socketio.emit('my_response',
                      {'data': 'Server generated event', 'count': count})


@app.route('/')
def index():
    return render_template('index.html', async_mode=socketio.async_mode)


@socketio.event
def my_event(message):
    session['receive_count'] = session.get('receive_count', 0) + 1
    emit('my_response',
         {'data': message['data'], 'count': session['receive_count']})


@socketio.event
def my_broadcast_event(message):
    session['receive_count'] = session.get('receive_count', 0) + 1
    emit('my_response',
         {'data': message['data'], 'count': session['receive_count']},
         broadcast=True)


@socketio.event
def join(message):
    join_room(message['room'])
    session['receive_count'] = session.get('receive_count', 0) + 1
    emit('my_response',
         {'data': 'In rooms: ' + ', '.join(rooms()),
          'count': session['receive_count']})
```

阅读排行榜

评论排行榜


```
@socketio.event
def leave(message):
    leave_room(message['room'])
    session['receive_count'] = session.get('receive_count', 0) + 1
    emit('my_response',
         {'data': 'In rooms: ' + ', '.join(rooms()),
          'count': session['receive_count']})


@socketio.on('close_room')
def on_close_room(message):
    session['receive_count'] = session.get('receive_count', 0) + 1
    emit('my_response', {'data': 'Room ' + message['room'] + ' is closing.',
                         'count': session['receive_count']},
         to=message['room'])
    close_room(message['room'])


@socketio.event
def my_room_event(message):
    session['receive_count'] = session.get('receive_count', 0) + 1
    emit('my_response',
         {'data': message['data'], 'count': session['receive_count']},
         to=message['room'])


@socketio.event
def disconnect_request():
    @copy_current_request_context
    def can_disconnect():
        disconnect()
```

推荐排行榜

1. 回首2018，展望2019(7)
2. 手把手教你搭建一个Elasticsearch集群(6)
3. Python文件读写（open()，close()，with open() as f...）(6)
4. python爬虫常见面试题（二）(4)
5. 关于学习编程的心得体会(3)

最新评论

1. Re:【websocket】小白快速上手flask-socketio

写的真好

--仙女的博客

2. Re:关于学习编程的心得体会

@逆水寒龙 是的，之前的工作用到过fastapi...

--天意凉

3. Re:关于学习编程的心得体会

```
    session['receive_count'] = session.get('receive_count', 0) + 1
    # for this emit we use a callback function
    # when the callback function is invoked we know that the message has been
    # received and it is safe to disconnect
    emit('my_response',
         {'data': 'Disconnected!', 'count': session['receive_count']},
         callback=can_disconnect)


@socketio.event
def my_ping():
    emit('my_pong')


@socketio.event
def connect():
    global thread
    with thread_lock:
        if thread is None:
            thread = socketio.start_background_task(background_thread)
    emit('my_response', {'data': 'Connected', 'count': 0})


@socketio.on('disconnect')
def test_disconnect():
    print('Client disconnected', request.sid)


if __name__ == '__main__':
    socketio.run(app, host='0.0.0.0', debug=True)
```

AI助手

复制

@kingdumpling 是的，会结合视频和文档来学习...

--天意凉

4. Re:关于学习编程的心得体会

@bananaplan 加油，日拱一卒！...

--天意凉

5. Re:关于学习编程的心得体会

换fastapi或者sanic，一起学啊

--逆水寒龙

（3）index.html代码

复制

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Flask-SocketIO Test</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js" integrity="sh
    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/3.0.4/socket.io.js" integrity='
    <script type="text/javascript" charset="utf-8">
        $(document).ready(function() {
            // Connect to the Socket.IO server.
            // The connection URL has the following format, relative to the current page:
            //     http[s]://<domain>:<port>[/<namespace>]
            var socket = io.connect('http://' + document.domain + ':' + location.port);

            // Event handler for new connections.
            // The callback function is invoked when a connection with the
            // server is established.
            socket.on('connect', function() {
                socket.emit('my_event', {data: 'I\'m connected!'});
            });

            // Event handler for server sent data.
            // The callback function is invoked whenever the server emits data
            // to the client. The data is then displayed in the "Received"
            // section of the page.
            socket.on('my_response', function(msg, cb) {
                $('#log').append('<br>' + $('<div/>').text('Received #' + msg.count + ': ' + msg.
                if (cb)
                    cb();
            });
```

```
// Interval function that tests message latency by sending a "ping"
// message. The server then responds with a "pong" message and the
// round trip time is measured.
var ping_pong_times = [];
var start_time;
window.setInterval(function() {
    start_time = (new Date).getTime();
    $('#transport').text(socket.io.engine.transport.name);
    socket.emit('my_ping');
}, 1000);

// Handler for the "pong" message. When the pong is received, the
// time from the ping is stored, and the average of the last 30
// samples is average and displayed.
socket.on('my_pong', function() {
    var latency = (new Date).getTime() - start_time;
    ping_pong_times.push(latency);
    ping_pong_times = ping_pong_times.slice(-30); // keep last 30 samples
    var sum = 0;
    for (var i = 0; i < ping_pong_times.length; i++)
        sum += ping_pong_times[i];
    $('#ping-pong').text(Math.round(10 * sum / ping_pong_times.length) / 10);
});

// Handlers for the different forms in the page.
// These accept data from the user and send it to the server in a
// variety of ways
$('form#emit').submit(function(event) {
    socket.emit('my_event', {data: $('#emit_data').val()});
    return false;
});
$('form#broadcast').submit(function(event) {
```

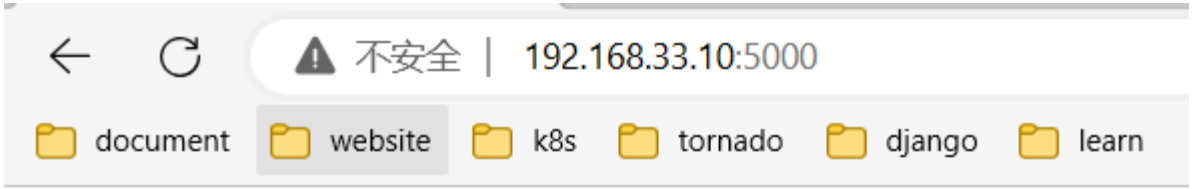
```
                socket.emit('my_broadcast_event', {data: $('#broadcast_data').val()});
                return false;
            });
            $('form#join').submit(function(event) {
                socket.emit('join', {room: $('#join_room').val()});
                return false;
            });
            $('form#leave').submit(function(event) {
                socket.emit('leave', {room: $('#leave_room').val()});
                return false;
            });
            $('form#send_room').submit(function(event) {
                socket.emit('my_room_event', {room: $('#room_name').val(), data: $('#room_data').
                return false;
            });
            $('form#close').submit(function(event) {
                socket.emit('close_room', {room: $('#close_room').val()});
                return false;
            });
            $('form#disconnect').submit(function(event) {
                socket.emit('disconnect_request');
                return false;
            });
        });
    </script>
</head>
<body>
    <h1>Flask-SocketIO Test</h1>
    <p>
      Async mode is: <b>{{ async_mode }}</b><br>
      Current transport is: <b><span id="transport"></span></b><br>
      Average ping/pong latency: <b><span id="ping-pong"></span>ms</b>
    </p>
```

```
    <h2>Send:</h2>
    <form id="emit" method="POST" action='#'>
        <input type="text" name="emit_data" id="emit_data" placeholder="Message">
        <input type="submit" value="Echo">
    </form>
    <form id="broadcast" method="POST" action='#'>
        <input type="text" name="broadcast_data" id="broadcast_data" placeholder="Message">
        <input type="submit" value="Broadcast">
    </form>
    <form id="join" method="POST" action='#'>
        <input type="text" name="join_room" id="join_room" placeholder="Room Name">
        <input type="submit" value="Join Room">
    </form>
    <form id="leave" method="POST" action='#'>
        <input type="text" name="leave_room" id="leave_room" placeholder="Room Name">
        <input type="submit" value="Leave Room">
    </form>
    <form id="send_room" method="POST" action='#'>
        <input type="text" name="room_name" id="room_name" placeholder="Room Name">
        <input type="text" name="room_data" id="room_data" placeholder="Message">
        <input type="submit" value="Send to Room">
    </form>
    <form id="close" method="POST" action="#">
        <input type="text" name="close_room" id="close_room" placeholder="Room Name">
        <input type="submit" value="Close Room">
    </form>
    <form id="disconnect" method="POST" action="#">
        <input type="submit" value="Disconnect">
    </form>
    <h2>Receive:</h2>
    <div id="log"></div>
```

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

复制

（4）运行app.py代码，浏览器访问5000端口，如下：



Flask-SocketIO Test

Async mode is: **eventlet**
Current transport is: **websocket**
Average ping/pong latency: **5.2ms**

Send:

Message Echo
Message Broadcast
Room Name Join Room
Room Name Leave Room
Room Name Message Send to Room
Room Name Close Room
Disconnect

Receive:

AI助手

Received #0: Connected

```
Received #1: I'm connected!
Received #1: Server generated event
Received #2: Server generated event
Received #3: Server generated event
Received #4: Server generated event
Received #5: Server generated event
```

（5）代码理解（最重要的部分！！！）

flask-socketio包的常用方法理解：

1. **socketio.on和socketio.event是等价**的，都是用来定义事件处理器（event handlers）的。区别是**.on的第一个参数是事件名称（event name），.event没有这个参数，而是使用被装饰的函数名作为事件名称。其他参数是一样的。**事件名称 **connect / disconnect / message / json** 都是SocketIO生成的特殊事件名，**任何其他的事件名都被视为自定义事件。**其他参数还有namespace（命名空间）。

2. **send和emit都被服务器用来向客户端发送消息。**send直接发送消息，emit需要指定事件和消息。**一般情况下，都是使用emit指定事件名发送消息。**emit的其他参数有：

   A. namespace（命名空间），和事件名配合使用。默认为"/"。

   B. broadcast（广播模式True/False），是否向所有客户端Client发送消息。

   C. to，通常为room_id，发送给指定房间的所有用户。

   D. callback（回调函数），指定回调函数，发送到另一端执行。

启动后的运行流程理解：

1. **启动时的初始运行流程。**客户端访问http://host:5000后，触发index.html里面的js代码，客户端执行了后，

```
var socket = io.connect('http://' + document.domain + ':' + location.port);
```

客户端和后台服务器建立了连接，注意，此时先触发服务器端的代码：

```
@socketio.event
def connect():
    global thread
    with thread_lock:
        if thread is None:
            thread = socketio.start_background_task(background_thread)
    emit('my_response', {'data': 'Connected', 'count': 0})
```

然后紧接着触发了客户端的代码：

```
socket.on('connect', function() {
                socket.emit('my_event', {data: 'I\'m connected!'});
            });
```

所以，浏览器请求的web页面 **Receive部分**：先是 **Received #0: Connected**，再是 **Received #1: I'm connected!**

2. **接下来，看看浏览器请求的web页面 Send部分：**

（1）echo：输入123，浏览器会向服务器端的my_event事件处理器发送数据{"data": 123}

```
socket.emit('my_event', {data: $('#emit_data').val()});
```

服务器端的my_event事件处理器为：

```
@socketio.event
def my_event(message):
    session['receive_count'] = session.get('receive_count', 0) + 1
    emit('my_response',
         {'data': message['data'], 'count': session['receive_count']})
```

可以看到，**服务器端在接收到数据后，又向客户端的my_response事件处理器发送数据。**在看看index.html里的my_response事件处理器是如何定义的：

```
socket.on('my_response', function(msg, cb) {
                $('#log').append('<br>' + $('<div/>').text('Received #' + msg.count + ': ' + msg.
                if (cb)
                    cb();
            });
```

最终，浏览器的web页面显示为 **Received #2: 123。通过这个例子，也充分展示了websocket的功能，服务端和客户端都主动可以向另一端发送数据。这是有别于http的。http协议只能客户端发起请求，服务端响应请求。服务端无法主动向客户端发送数据。**

（2）broadcast暂时不说了。

（3）Join Room：这个和Leave Room是成对使用的。就像一个聊天室一样，加入指定聊天室后，当执行Send to Room，就可以接收这个房间内的所有消息。

（4）Close Room：关闭房间

（5）Disconnect：客户端主动断开连接，客户端触发服务端的disconnect_request事件处理器，

```
# 客户端
socket.emit('disconnect_request');

# 服务端
@socketio.event
def disconnect_request():
    @copy_current_request_context
    def can_disconnect():
        disconnect()
    session['receive_count'] = session.get('receive_count', 0) + 1
    # for this emit we use a callback function
    # when the callback function is invoked we know that the message has been
    # received and it is safe to disconnect
    emit('my_response',
         {'data': 'Disconnected!', 'count': session['receive_count']},
         callback=can_disconnect)
```

服务端收到请求后，会向客户端的my_response事件处理器发送数据，同时发送一个callback回调函数can_disconnect，让客户端执行该函数。

最终浏览器的页面显示：**Received #2: Disconnected!**

三、写在最后

至此，你应该已经对使用flask-socketio库有了基本的认识了。如果还有不了解的，可以留言交流。

在生产环境中，还需要添加异常处理，比如socketio.on_error()和socketio.on_error_default()。

本文只是入门使用教程，感兴趣的话请大家自行查文档深入理解。

附上**官方教程链接**:

1. https://blog.miguelgrinberg.com/post/easy-websockets-with-flask-and-gevent

2. https://flask-socketio.readthedocs.io/en/latest/index.html

合集: flask框架

标签: web开发 , websocket

好文要顶 关注我 收藏该文 微信分享

天意凉
粉丝 - 43 关注 - 11

+加关注

1 推荐 0 反对

升级成为会员

« 上一篇: 关于学习编程的心得体会

posted @ 2023-07-11 19:47 天意凉 阅读(5560) 评论(1) 编辑 收藏 举报

刷新页面 返回顶部

登录后才能查看或发表评论，立即 登录 或者 逛逛 博客园首页

AI助手

编辑推荐：

· 在.NET Core中使用异步多线程高效率的处理大量数据
· 聊一聊 C#前台线程 如何阻塞程序退出
· 几种数据库优化技巧
· 聊一聊坑人的 C# MySql.Data SDK
· 使用 .NET Core 实现一个自定义日志记录器

阅读排行：

· 字节豆包，来园广告
· 我用cursor, 半就开发了一个手机壁纸小程序，真的太强了
· 为什么推荐在 .NET 中使用 YAML 配置文件
· 订单超时自动取消，我们是这样做的。。。
· 在 .NET Core 中使用 Channel 实现高效率的多步骤处理大量数据