你真的了解 WebSocket 吗？

2017/08/01 · [实践项目](http://python.jobbole.com/category/project/) · [websocket](http://python.jobbole.com/tag/websocket/)

分享到：

原文出处： [Mr.Seven](http://www.cnblogs.com/wupeiqi/p/6558766.html)

WebSocket协议是基于TCP的一种新的协议。WebSocket最初在HTML5规范中被引用为TCP连接，作为基于TCP的套接字API的占位符。它实现了浏览器与服务器全双工(full-duplex)通信。其本质是保持TCP连接，在浏览器和服务端通过Socket进行通信。

本文将使用Python编写Socket服务端，一步一步分析请求过程！！！

**1. 启动服务端**

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import socket  sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  sock.bind(('127.0.0.1', 8002))  sock.listen(5)  # 等待用户连接  conn, address = sock.accept()  ...  ...  ... |

启动Socket服务器后，等待用户【连接】，然后进行收发数据。

**2. 客户端连接**

Python



|  |  |
| --- | --- |
| 1  2  3  4 | <script type="text/javascript">      var socket = new WebSocket("ws://127.0.0.1:8002/xxoo");      ...  </script> |

当客户端向服务端发送连接请求时，不仅连接还会发送【握手】信息，并等待服务端响应，至此连接才创建成功！

**3. 建立连接【握手】**

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import socket    sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  sock.bind(('127.0.0.1', 8002))  sock.listen(5)  # 获取客户端socket对象  conn, address = sock.accept()  # 获取客户端的【握手】信息  data = conn.recv(1024)  ...  ...  ...  conn.send('响应【握手】信息') |

请求和响应的【握手】信息需要遵循规则：

* 从请求【握手】信息中提取 Sec-WebSocket-Key
* 利用magic\_string 和 Sec-WebSocket-Key 进行hmac1加密，再进行base64加密
* 将加密结果响应给客户端

*注：magic string为：258EAFA5-E914-47DA-95CA-C5AB0DC85B11*

请求【握手】信息为：

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | GET /chatsocket HTTP/1.1  Host: 127.0.0.1:8002  Connection: Upgrade  Pragma: no-cache  Cache-Control: no-cache  Upgrade: websocket  Origin: http://localhost:63342  Sec-WebSocket-Version: 13  Sec-WebSocket-Key: mnwFxiOlctXFN/DeMt1Amg==  Sec-WebSocket-Extensions: permessage-deflate; client\_max\_window\_bits  ...  ... |

提取Sec-WebSocket-Key值并加密：

Python



|  |  |
| --- | --- |
| 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | import socket  import base64  import hashlib    def get\_headers(data):      """      将请求头格式化成字典      :param data:      :return:      """      header\_dict = {}      data = str(data, encoding='utf-8')        for i in data.split('\r\n'):          print(i)      header, body = data.split('\r\n\r\n', 1)      header\_list = header.split('\r\n')      for i in range(0, len(header\_list)):          if i == 0:              if len(header\_list[i].split(' ')) == 3:                  header\_dict['method'], header\_dict['url'], header\_dict['protocol'] = header\_list[i].split(' ')          else:              k, v = header\_list[i].split(':', 1)              header\_dict[k] = v.strip()      return header\_dict      sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  sock.bind(('127.0.0.1', 8002))  sock.listen(5)    conn, address = sock.accept()  data = conn.recv(1024)  headers = get\_headers(data) # 提取请求头信息  # 对请求头中的sec-websocket-key进行加密  response\_tpl = "HTTP/1.1 101 Switching Protocols\r\n" \        "Upgrade:websocket\r\n" \        "Connection: Upgrade\r\n" \        "Sec-WebSocket-Accept: %s\r\n" \        "WebSocket-Location: ws://%s%s\r\n\r\n"  magic\_string = '258EAFA5-E914-47DA-95CA-C5AB0DC85B11'  value = headers['Sec-WebSocket-Key'] + magic\_string  ac = base64.b64encode(hashlib.sha1(value.encode('utf-8')).digest())  response\_str = response\_tpl % (ac.decode('utf-8'), headers['Host'], headers['url'])  # 响应【握手】信息  conn.send(bytes(response\_str, encoding='utf-8'))  ...  ...  ... |

**4.客户端和服务端收发数据**

客户端和服务端传输数据时，需要对数据进行【封包】和【解包】。客户端的JavaScript类库已经封装【封包】和【解包】过程，但Socket服务端需要手动实现。

第一步：获取客户端发送的数据【解包】

Python

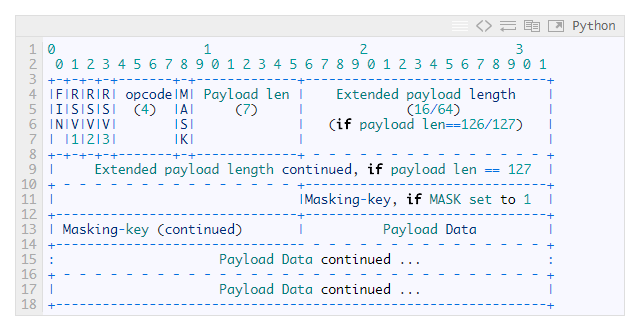


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | info = conn.recv(8096)        payload\_len = info[1] & 127      if payload\_len == 126:          extend\_payload\_len = info[2:4]          mask = info[4:8]          decoded = info[8:]      elif payload\_len == 127:          extend\_payload\_len = info[2:10]          mask = info[10:14]          decoded = info[14:]      else:          extend\_payload\_len = None          mask = info[2:6]          decoded = info[6:]        bytes\_list = bytearray()      for i in range(len(decoded)):          chunk = decoded[i] ^ mask[i % 4]          bytes\_list.append(chunk)      body = str(bytes\_list, encoding='utf-8')      print(body)    基于Python实现解包过程（未实现长内容） |

解包详细过程：

Python





第二步：向客户端发送数据【封包】

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | def send\_msg(conn, msg\_bytes):      """      WebSocket服务端向客户端发送消息      :param conn: 客户端连接到服务器端的socket对象,即： conn,address = socket.accept()      :param msg\_bytes: 向客户端发送的字节      :return:      """      import struct        token = b"\x81"      length = len(msg\_bytes)      if length < 126:          token += struct.pack("B", length)      elif length <= 0xFFFF:          token += struct.pack("!BH", 126, length)      else:          token += struct.pack("!BQ", 127, length)        msg = token + msg\_bytes      conn.send(msg)      return True |

**5. 基于Python实现简单示例**

a. 基于Python socket实现的WebSocket服务端：

Python



|  |  |
| --- | --- |
| 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103 | #!/usr/bin/env python  # -\*- coding:utf-8 -\*-  import socket  import base64  import hashlib      def get\_headers(data):      """      将请求头格式化成字典      :param data:      :return:      """      header\_dict = {}      data = str(data, encoding='utf-8')        header, body = data.split('\r\n\r\n', 1)      header\_list = header.split('\r\n')      for i in range(0, len(header\_list)):          if i == 0:              if len(header\_list[i].split(' ')) == 3:                  header\_dict['method'], header\_dict['url'], header\_dict['protocol'] = header\_list[i].split(' ')          else:              k, v = header\_list[i].split(':', 1)              header\_dict[k] = v.strip()      return header\_dict      def send\_msg(conn, msg\_bytes):      """      WebSocket服务端向客户端发送消息      :param conn: 客户端连接到服务器端的socket对象,即： conn,address = socket.accept()      :param msg\_bytes: 向客户端发送的字节      :return:      """      import struct        token = b"\x81"      length = len(msg\_bytes)      if length < 126:          token += struct.pack("B", length)      elif length <= 0xFFFF:          token += struct.pack("!BH", 126, length)      else:          token += struct.pack("!BQ", 127, length)        msg = token + msg\_bytes      conn.send(msg)      return True      def run():      sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)      sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)      sock.bind(('127.0.0.1', 8003))      sock.listen(5)        conn, address = sock.accept()      data = conn.recv(1024)      headers = get\_headers(data)      response\_tpl = "HTTP/1.1 101 Switching Protocols\r\n" \                     "Upgrade:websocket\r\n" \                     "Connection:Upgrade\r\n" \                     "Sec-WebSocket-Accept:%s\r\n" \                     "WebSocket-Location:ws://%s%s\r\n\r\n"        value = headers['Sec-WebSocket-Key'] + '258EAFA5-E914-47DA-95CA-C5AB0DC85B11'      ac = base64.b64encode(hashlib.sha1(value.encode('utf-8')).digest())      response\_str = response\_tpl % (ac.decode('utf-8'), headers['Host'], headers['url'])      conn.send(bytes(response\_str, encoding='utf-8'))        while True:          try:              info = conn.recv(8096)          except Exception as e:              info = None          if not info:              break          payload\_len = info[1] & 127          if payload\_len == 126:              extend\_payload\_len = info[2:4]              mask = info[4:8]              decoded = info[8:]          elif payload\_len == 127:              extend\_payload\_len = info[2:10]              mask = info[10:14]              decoded = info[14:]          else:              extend\_payload\_len = None              mask = info[2:6]              decoded = info[6:]            bytes\_list = bytearray()          for i in range(len(decoded)):              chunk = decoded[i] ^ mask[i % 4]              bytes\_list.append(chunk)          body = str(bytes\_list, encoding='utf-8')          send\_msg(conn,body.encode('utf-8'))        sock.close()    if \_\_name\_\_ == '\_\_main\_\_':      run() |

b. 利用JavaScript类库实现客户端

Python



|  |  |
| --- | --- |
| 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | <!DOCTYPE html>  <html>  <head lang="en">      <meta charset="UTF-8">      <title></title>  </head>  <body>      <div>          <input type="text" id="txt"/>          <input type="button" id="btn" value="提交" onclick="sendMsg();"/>          <input type="button" id="close" value="关闭连接" onclick="closeConn();"/>      </div>      <div id="content"></div>    <script type="text/javascript">      var socket = new WebSocket("ws://127.0.0.1:8003/chatsocket");        socket.onopen = function () {          /\* 与服务器端连接成功后，自动执行 \*/            var newTag = document.createElement('div');          newTag.innerHTML = "【连接成功】";          document.getElementById('content').appendChild(newTag);      };        socket.onmessage = function (event) {          /\* 服务器端向客户端发送数据时，自动执行 \*/          var response = event.data;          var newTag = document.createElement('div');          newTag.innerHTML = response;          document.getElementById('content').appendChild(newTag);      };        socket.onclose = function (event) {          /\* 服务器端主动断开连接时，自动执行 \*/          var newTag = document.createElement('div');          newTag.innerHTML = "【关闭连接】";          document.getElementById('content').appendChild(newTag);      };        function sendMsg() {          var txt = document.getElementById('txt');          socket.send(txt.value);          txt.value = "";      }      function closeConn() {          socket.close();          var newTag = document.createElement('div');          newTag.innerHTML = "【关闭连接】";          document.getElementById('content').appendChild(newTag);      }    </script>  </body>  </html> |

**6. 基于Tornado框架实现Web聊天室**

Tornado是一个支持WebSocket的优秀框架，其内部原理正如1~5步骤描述，当然Tornado内部封装功能更加完整。 以下是基于Tornado实现的聊天室示例：

Python



|  |  |
| --- | --- |
| 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | #!/usr/bin/env python  # -\*- coding:utf-8 -\*-  import uuid  import json  import tornado.ioloop  import tornado.web  import tornado.websocket      class IndexHandler(tornado.web.RequestHandler):      def get(self):          self.render('index.html')      class ChatHandler(tornado.websocket.WebSocketHandler):      # 用户存储当前聊天室用户      waiters = set()      # 用于存储历时消息      messages = []        def open(self):          """          客户端连接成功时，自动执行          :return:          """          ChatHandler.waiters.add(self)          uid = str(uuid.uuid4())          self.write\_message(uid)            for msg in ChatHandler.messages:              content = self.render\_string('message.html', \*\*msg)              self.write\_message(content)        def on\_message(self, message):          """          客户端连发送消息时，自动执行          :param message:          :return:          """          msg = json.loads(message)          ChatHandler.messages.append(message)            for client in ChatHandler.waiters:              content = client.render\_string('message.html', \*\*msg)              client.write\_message(content)        def on\_close(self):          """          客户端关闭连接时，，自动执行          :return:          """          ChatHandler.waiters.remove(self)      def run():      settings = {          'template\_path': 'templates',          'static\_path': 'static',      }      application = tornado.web.Application([          (r"/", IndexHandler),          (r"/chat", ChatHandler),      ], \*\*settings)      application.listen(8888)      tornado.ioloop.IOLoop.instance().start()      if \_\_name\_\_ == "\_\_main\_\_":      run()    app.py |

Python



|  |  |
| --- | --- |
| 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  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <title>Python聊天室</title>  </head>  <body>      <div>          <input type="text" id="txt"/>          <input type="button" id="btn" value="提交" onclick="sendMsg();"/>          <input type="button" id="close" value="关闭连接" onclick="closeConn();"/>      </div>      <div id="container" style="border: 1px solid #dddddd;margin: 20px;min-height: 500px;">        </div>        <script src="/static/jquery-2.1.4.min.js"></script>      <script type="text/javascript">          $(function () {              wsUpdater.start();          });            var wsUpdater = {              socket: null,              uid: null,              start: function() {                  var url = "ws://127.0.0.1:8888/chat";                  wsUpdater.socket = new WebSocket(url);                  wsUpdater.socket.onmessage = function(event) {                      console.log(event);                      if(wsUpdater.uid){                          wsUpdater.showMessage(event.data);                      }else{                          wsUpdater.uid = event.data;                      }                  }              },              showMessage: function(content) {                  $('#container').append(content);              }          };            function sendMsg() {              var msg = {                  uid: wsUpdater.uid,                  message: $("#txt").val()              };              wsUpdater.socket.send(JSON.stringify(msg));          }    </script>    </body>  </html>    index.html |

[示例源码下载](http://files.cnblogs.com/files/wupeiqi/WebChat.zip)

**参考文献**：https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\_API/Writing\_WebSocket\_servers