

## 18 | 新特性：Tomcat如何支持WebSocket?

2019-06-20 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)



**讲述：李号双**

时长 11:00 大小 8.83M



我们知道 HTTP 协议是“请求 - 响应”模式，浏览器必须先发请求给服务器，服务器才会响应这个请求。也就是说，服务器不会主动发送数据给浏览器。

对于实时性要求比较高高的应用，比如在线游戏、股票基金实时报价和在线协同编辑等，浏览器需要实时显示服务器上最新的数据，因此出现了 Ajax 和 Comet 技术。Ajax 本质上还是轮询，而 Comet 是在 HTTP 长连接的基础上做了一些 hack，但是它们的实时性不高，另外频繁的请求会给服务器带来压力，也会浪费网络流量和带宽。于是 HTML5 推出了 WebSocket 标准，使得浏览器和服务器之间任何一方都可以主动发消息给对方，这样服务器有新数据时可以主动推送给浏览器。

今天我会介绍 WebSocket 的工作原理，以及作为服务器端的 Tomcat 是如何支持 WebSocket 的。更重要的是，希望你在学完之后可以灵活地选用 WebSocket 技术来解决实际工作中的问题。

## WebSocket 工作原理

WebSocket 的名字里带有 Socket，那 Socket 是什么呢？网络上的两个程序通过一个双向链路进行通信，这个双向链路的一端称为一个 Socket。一个 Socket 对应一个 IP 地址和端口号，应用程序通常通过 Socket 向网络发出请求或者应答网络请求。Socket 不是协议，它其实是对 TCP/IP 协议层抽象出来的 API。

但 WebSocket 不是一套 API，跟 HTTP 协议一样，WebSocket 也是一个应用层协议。为了跟现有的 HTTP 协议保持兼容，它通过 HTTP 协议进行一次握手，握手之后数据就直接从 TCP 层的 Socket 传输，就与 HTTP 协议无关了。浏览器发给服务端的请求会带上跟 WebSocket 有关的请求头，比如 `Connection: Upgrade` 和 `Upgrade: websocket`。

**Connection: Upgrade**

**Host:** localhost:8080

**Origin:** http://localhost:8080

**Pragma:** no-cache

**Sec-WebSocket-Extensions:** permessage-deflate; client\_max\_window\_bits

**Sec-WebSocket-Key:** oZxS2fjcIXd/vEwycKaHZA==

**Sec-WebSocket-Version:** 13

**Upgrade: websocket**

如果服务器支持 WebSocket，同样会在 HTTP 响应里加上 WebSocket 相关的 HTTP 头部。

**Connection: upgrade**

**Date:** Thu, 04 Apr 2019 03:57:41 GMT

**Sec-WebSocket-Accept:** A70nHM3MTE8jaArD8fREfIXzmH=

**Sec-WebSocket-Extensions:** permessage-deflate; client\_max\_window\_bits=15

**Upgrade: websocket**

---

这样 WebSocket 连接就建立好了，接下来 WebSocket 的数据传输会以 frame 形式传输，会将一条消息分为几个 frame，按照先后顺序传输出去。这样做的好处有：

大数据的传输可以分片传输，不用考虑数据大小的问题。


和 HTTP 的 chunk 一样，可以边生成数据边传输，提高传输效率。

## Tomcat 如何支持 WebSocket

在讲 Tomcat 如何支持 WebSocket 之前，我们先来开发一个简单的聊天室程序，需求是：用户可以通过浏览器加入聊天室、发送消息，聊天室的其他人都可以收到消息。

### WebSocket 聊天室程序

浏览器端 JavaScript 核心代码如下：

 复制代码

```
1 var Chat = {};  
2 Chat.socket = null;  
3 Chat.connect = (function(host) {  
4  
5     // 判断当前浏览器是否支持 WebSocket  
6     if ('WebSocket' in window) {  
7         // 如果支持则创建 WebSocket JS 类
```


```
8         Chat.socket = new WebSocket(host);
9     } else if ('MozWebSocket' in window) {
10         Chat.socket = new MozWebSocket(host);
11     } else {
12         Console.log('WebSocket is not supported by this
13         return;
14     }
15
16     // 回调函数，当和服务器的 WebSocket 连接建立起来后，浏
17     Chat.socket.onopen = function () {
18         Console.log('Info: WebSocket connection opened.
19         document.getElementById('chat').onkeydown = fun
20             if (event.keyCode == 13) {
21                 Chat.sendMessage();
22             }
23         };
24     };
25
26     // 回调函数，当和服务器的 WebSocket 连接关闭后，浏览器
27     Chat.socket.onclose = function () {
28         document.getElementById('chat').onkeydown = nul
29         Console.log('Info: WebSocket closed.');
```



```
30     };
31
32     // 回调函数，当服务器有新消息发送到浏览器，浏览器会回调
33     Chat.socket.onmessage = function (message) {
34         Console.log(message.data);
35     };
36 });
```

上面的代码实现逻辑比较清晰，就是创建一个 WebSocket JavaScript 对象，然后实现了几个回调方法：onopen、

onclose 和 onmessage。当连接建立、关闭和有新消息时，浏览器会负责调用这些回调方法。我们再来看服务器端 Tomcat 的实现代码：

 复制代码

```
1 //Tomcat 端的实现类加上 @ServerEndpoint 注解，里面的 valu
2 @ServerEndpoint(value = "/websocket/chat")
3 public class ChatEndpoint {
4
5     private static final String GUEST_PREFIX = "Guest";
6
7     // 记录当前有多少个用户加入到了聊天室，它是 static 全局
8     private static final AtomicInteger connectionIds =
9
10    // 每个用户用一个 CharAnnotation 实例来维护，请你注意
11    private static final Set<ChatEndpoint> connections
12        new CopyOnWriteArraySet<>();
13
14    private final String nickname;
15    private Session session;
16
17    public ChatEndpoint() {
18        nickname = GUEST_PREFIX + connectionIds.getAndIncr
19    }
20
21    // 新连接到达时，Tomcat 会创建一个 Session，并回调这个
22    @OnOpen
23    public void start(Session session) {
24        this.session = session;
25        connections.add(this);
26        String message = String.format("* %s %s", nickr
27        broadcast(message);
28    }
```

```

29
30 // 浏览器关闭连接时，Tomcat 会回调这个函数
31 @OnClose
32 public void end() {
33     connections.remove(this);
34     String message = String.format("* %s %s",
35         nickname, "has disconnected.");
36     broadcast(message);
37 }
38
39 // 浏览器发送消息到服务器时，Tomcat 会回调这个函数
40 @OnMessage
41 public void incoming(String message) {
42     // Never trust the client
43     String filteredMessage = String.format("%s: %s"
44         nickname, HTMLFilter.filter(message.toS
45     broadcast(filteredMessage);
46 }
47
48 //Websocket 连接出错时，Tomcat 会回调这个函数
49 @OnError
50 public void onError(Throwable t) throws Throwable {
51     log.error("Chat Error: " + t.toString(), t);
52 }
53
54 // 向聊天室中的每个用户广播消息
55 private static void broadcast(String msg) {
56     for (ChatAnnotation client : connections) {
57         try {
58             synchronized (client) {
59                 client.session.getBasicRemote().ser
60             }
61         } catch (IOException e) {
62             ...
63         }

```

```
64         }  
65     }  
66 }
```

根据 Java WebSocket 规范的规定，Java WebSocket 应用程序由一系列的 WebSocket Endpoint 组成。**Endpoint 是一个 Java 对象，代表 WebSocket 连接的一端，就好像处理 HTTP 请求的 Servlet 一样，你可以把它看作是处理 WebSocket 消息的接口。**跟 Servlet 不同的地方在于，Tomcat 会给每一个 WebSocket 连接创建一个 Endpoint 实例。你可以通过两种方式定义和实现 Endpoint。

第一种方法是编程式的，就是编写一个 Java 类继承 `javax.websocket.Endpoint`，并实现它的 `onOpen`、`onClose` 和 `onError` 方法。这些方法跟 Endpoint 的生命周期有关，Tomcat 负责管理 Endpoint 的生命周期并调用这些方法。并且当浏览器连接到一个 Endpoint 时，Tomcat 会给这个连接创建一个唯一的 `Session (javax.websocket.Session)`。Session 在 WebSocket 连接握手成功之后创建，并在连接关闭时销毁。当触发 Endpoint 各个生命周期事件时，Tomcat 会将当前 Session 作为参数传给 Endpoint 的回调方法，因此一个 Endpoint 实例对应一个 Session，我们通过 Session



中添加 `MessageHandler` 消息处理器来接收消息，`MessageHandler` 中定义了 `onMessage` 方法。在这里 **Session 的本质是对 Socket 的封装，Endpoint 通过它与浏览器通信。**

第二种定义 `Endpoint` 的方法是注解式的，也就是上面的聊天室程序例子中用到的方式，即实现一个业务类并给它添加 `WebSocket` 相关的注解。首先我们注意到

```
@ServerEndpoint(value = "/websocket/chat")
```

注解，它表明当前业务类 `ChatEndpoint` 是一个实现了 `WebSocket` 规范的 `Endpoint`，并且注解的 `value` 值表明 `ChatEndpoint` 映射的 URL 是 `/websocket/chat`。我们还看到 `ChatEndpoint` 类中有 `@OnOpen`、`@OnClose`、`@OnError` 和在 `@OnMessage` 注解的方法，从名字你就知道它们的功能是什么。


对于程序员来说，其实我们只需要专注具体的 `Endpoint` 的实现，比如在上面聊天室的例子中，为了方便向所有人群发消息，`ChatEndpoint` 在内部使用了一个全局静态的集合 `CopyOnWriteArraySet` 来维护所有的 `ChatEndpoint` 实例，因为每一个 `ChatEndpoint` 实例对应一个 `WebSocket` 连接，也就是代表了一个加入聊天室的用户。**当某个 `ChatEndpoint` 实例收到来自浏览器的消息时，这个**

**ChatEndpoint 会向集合中其他 ChatEndpoint 实例背后的 WebSocket 连接推送消息。**


那么这个过程中，Tomcat 主要做了哪些事情呢？简单来说就是两件事情：**Endpoint 加载和 WebSocket 请求处理**。下面我分别来详细说说 Tomcat 是如何做这两件事情的。

## WebSocket 加载

Tomcat 的 WebSocket 加载是通过 SCI 机制完成的。SCI 全称 ServletContainerInitializer，是 Servlet 3.0 规范中定义的用来**接收 Web 应用启动事件的接口**。那为什么要监听 Servlet 容器的启动事件呢？因为这样我们有机会在 Web 应用启动时做一些初始化工作，比如 WebSocket 需要扫描和加载 Endpoint 类。SCI 的使用也比较简单，将实现 ServletContainerInitializer 接口的类增加 HandlesTypes 注解，并且在注解内指定的一系列类和接口集合。比如 Tomcat 为了扫描和加载 Endpoint 而定义的 SCI 类如下：

 复制代码

```
1 @HandlesTypes({ServerEndpoint.class, ServerApplicationC
2 public class WsSci implements ServletContainerInitializ
3
4     public void onStartup(Set<Class<?>> clazzes, ServletC
5     ...
6 }
```

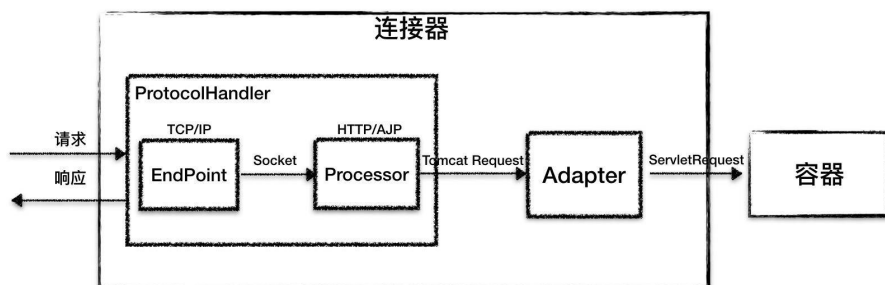


一旦定义好了 SCI, Tomcat 在启动阶段扫描类时, 会将 HandlesTypes 注解中指定的类都扫描出来, 作为 SCI 的 onStartUp 方法的参数, 并调用 SCI 的 onStartUp 方法。注意到 WsSci 的 HandlesTypes 注解中定义了 `ServerEndpoint.class`、`ServerApplicationConfig.class` 和 `Endpoint.class`, 因此在 Tomcat 的启动阶段会将这些类的类实例（注意不是对象实例）传递给 WsSci 的 onStartUp 方法。那么 WsSci 的 onStartUp 方法又做了什么事呢？

它会构造一个 `WebSocketContainer` 实例, 你可以把 `WebSocketContainer` 理解成一个专门处理 `WebSocket` 请求的**Endpoint 容器**。也就是说 Tomcat 会把扫描到的 `Endpoint` 子类和添加了注解 `@ServerEndpoint` 的类注册到这个容器中, 并且这个容器还维护了 URL 到 `Endpoint` 的映射关系, 这样通过请求 URL 就能找到具体的 `Endpoint` 来处理 `WebSocket` 请求。

## WebSocket 请求处理

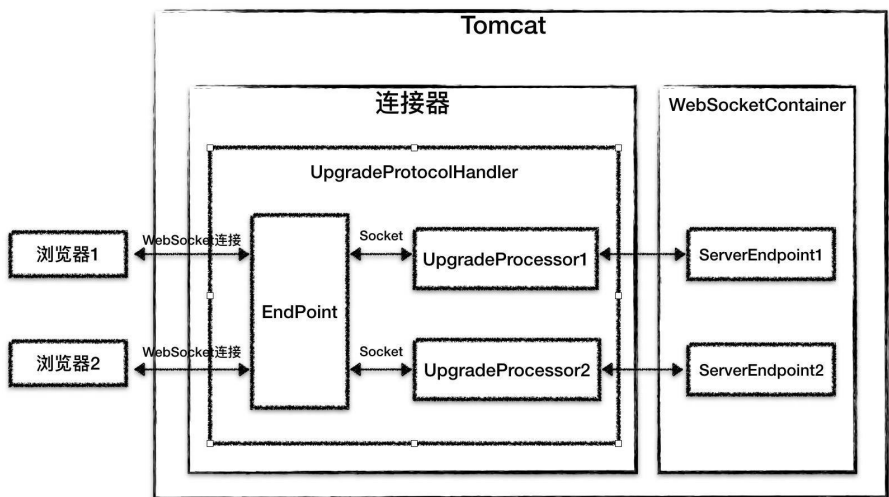
在讲 WebSocket 请求处理之前，我们先来回顾一下 Tomcat 连接器的组件图。



你可以看到 Tomcat 用 ProtocolHandler 组件屏蔽应用层协议的差异，其中 ProtocolHandler 中有两个关键组件：Endpoint 和 Processor。需要注意，这里的 Endpoint 跟上文提到的 WebSocket 中的 Endpoint 完全是两回事，连接器中的 Endpoint 组件用来处理 I/O 通信。WebSocket 本质就是一个应用层协议，因此不能用 HttpProcessor 来处理 WebSocket 请求，而要用专门 Processor 来处理，而在 Tomcat 中这样的 Processor 叫作 UpgradeProcessor。

为什么叫 Upgrade Processor 呢？这是因为 Tomcat 是将 HTTP 协议升级成 WebSocket 协议的，我们知道 WebSocket 是通过 HTTP 协议来进行握手的，因此当 WebSocket 的握手请求到来时，HttpProtocolHandler 首先接收到这个请求，在处理这个 HTTP 请求时，Tomcat 通

过一个特殊的 Filter 判断该当前 HTTP 请求是否是一个 WebSocket Upgrade 请求（即包含 Upgrade: websocket 的 HTTP 头信息），如果是，则在 HTTP 响应里添加 WebSocket 相关的响应头信息，并进行协议升级。具体来说就是用 UpgradeProtocolHandler 替换当前的 HttpProtocolHandler，相应的，把当前 Socket 的 Processor 替换成 UpgradeProcessor，同时 Tomcat 会创建 WebSocket Session 实例和 Endpoint 实例，并跟当前的 WebSocket 连接——对应起来。这个 WebSocket 连接不会立即关闭，并且在请求处理中，不再使用原有的 HttpProcessor，而是用专门的 UpgradeProcessor，UpgradeProcessor 最终会调用相应的 Endpoint 实例来处理请求。下面我们通过一张图来理解一下。



你可以看到，Tomcat 对 WebSocket 请求的处理没有经过 Servlet 容器，而是通过 UpgradeProcessor 组件直接把请求发到 ServerEndpoint 实例，并且 Tomcat 的 WebSocket 实现不需要关注具体 I/O 模型的细节，从而实现了与具体 I/O 方式的解耦。

## 本期精华

WebSocket 技术实现了 Tomcat 与浏览器的双向通信，Tomcat 可以主动向浏览器推送数据，可以用来实现对数据实时性要求比较高的应用。这需要浏览器和 Web 服务器同时支持 WebSocket 标准，Tomcat 启动时通过 SCI 技术来扫描和加载 WebSocket 的处理类 ServerEndpoint，并且建立起了 URL 到 ServerEndpoint 的映射关系。

当第一个 WebSocket 请求到达时，Tomcat 将 HTTP 协议升级成 WebSocket 协议，并将该 Socket 连接的 Processor 替换成 UpgradeProcessor。这个 Socket 不会立即关闭，对接下来的请求，Tomcat 通过 UpgradeProcessor 直接调用相应的 ServerEndpoint 来处理。

今天我讲了可以通过两种方式来开发 WebSocket 应用，一种是继承 `javax.websocket.Endpoint`，另一种通过

WebSocket 相关的注解。其实你还可以通过 Spring 来实现 WebSocket 应用，有兴趣的话你可以去研究一下 Spring WebSocket 的原理。

## 课后思考

今天我举的聊天室的例子实现的是群发消息，如果要向某个特定用户发送消息，应该怎么做呢？

不知道今天的内容你消化得如何？如果还有疑问，请大胆的在留言区提问，也欢迎你把你的课后思考和心得记录下来，与我和其他同学一起讨论。如果你觉得今天有所收获，欢迎你把它分享给你的朋友。



## 深入拆解 Tomcat & Jetty

从源码角度深度探索 Java 中间件

李号双

eBay 技术主管



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 17 | Executor组件：Tomcat如何扩展Java线程池？

下一篇 19 | 比较：Jetty的线程策略EatWhatYouKill

## 精选留言 (16)

写留言



QQ怪

2019-06-20

之前做过相关websocket服务的相关设计，只要把登陆对象用户标识作为key，websocketsessionId作为value缓存成map集合，通过key找到websocketsessionId发送即可，利用的是spring websocket，但我有一点不太清楚的是：我听说单机tomcat能容纳5万客户端，我不知...

作者回复: 5万感觉是Linux上一个进程打开的文件数限制，Linux把Socket也看作是一个文件。

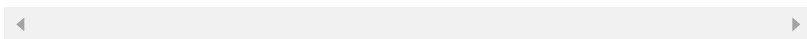
并发数没上去要看资源瓶颈在哪，内存，CPU或者带宽，一般大量链接需要耗费内存，还有就是Tomcat的最大连接数和最大线程数设置大了没有。

你的Websocket应用访问了全局变量就有并发问题，



Tomcat层面不会有全局锁的瓶颈。

你说的并发是指线程安全吗，要看websocket

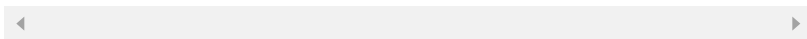


**吃饭饭**

2019-06-20

为了跟现有的 HTTP 协议保持兼容，它通过 HTTP 协议进行一次握手，这里不应该是三次握手吗老师？不太明白这里，求讲解

作者回复: 这里的握手是指应用协议层，不是tcp层。握手的时候tcp连接已经建立，就是http请求里带有websocket的请求头，服务端回复也带有websocket的响应头。

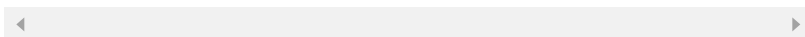


**foo**

2019-06-20

老师，一直不太理解注解的作用及原理，在网上查到注解的目的是标识源代码元素的元数据，是否是源码中通过反射检测到注解后，实现了此注解的实际动作？如何查看此注解的实际动作呢？（描述有点乱，见谅）

作者回复: 是的, 查找注解的实际动作可以在代码中全文搜索注解的关键字



**z.l**

2019-06-23

希望下一本能出《深入拆解netty》😁



**-W.LI-**

2019-06-23

老师好!我有个问题可能有点蠢。spring对websocket有支持, 还有STOMP这种。文中说websocket不是运行在severlet容器上的。spring上下文不是servlet容易下的一个子容器么? 没有servlet容器的话, 这个spring上下文注册去哪了啊?



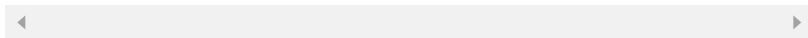
**Geek\_28b75e**

2019-06-21

老师, 测试环境遇到一个问题, 本项目使用springcloud, 网管使用zuul。我在网关路由之前的过滤

器添加了一个请求头，添加之后对于一般接口的路由转发是正常的，其中对于文件上传接口，我断点查看，需要好几秒才转发过去，并且报异常， ...

作者回复: 建议抓包分析



**Cy190622**

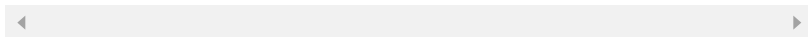
2019-06-20

老师，晚上好。

请教一下，

- 1.每节都有课后思考题，老师能否在下一节安排解答一下，或者是安排统一的解答章节嘛？
- 2.netty现在使用很流行。常用在生产上，咱们的课程是...

作者回复: 后面会有统一的答疑篇，Netty In Action 不错。

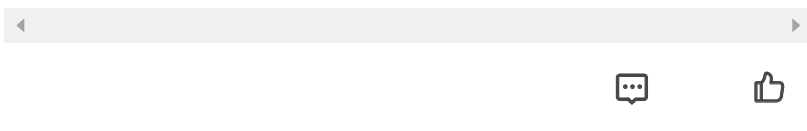


**杨俊**

2019-06-20

tomcat nodejs在websocket性能上哪个好呢

作者回复: netty最好：)



**Geek\_9d8c59**

2019-06-20

java版本的websocket客户端，websocket如果是分frame传输，不在意数据大小，为什么我把图片转换成base64，使用同步发送方法，会触发连接关闭。而把session的string消息类型的缓冲区设置的足够大之后，问题才消失。而后面在我加入了心跳包，重连机制后，当...

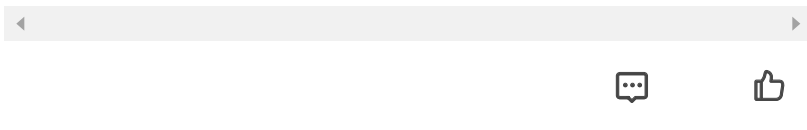


**andy**

2019-06-20

老师您好，我有个tomcat spring mvc应用，需要提供一个接口供socket连接，我可以直接在应用服务层用netty编写一个监听端口来实现吗

作者回复: 可以的，端口不冲突就行

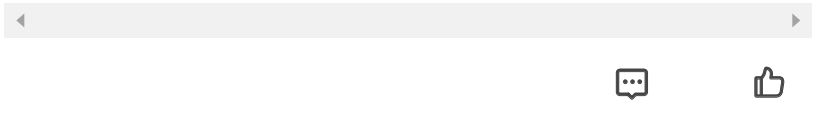


**咸鱼**

2019-06-20

老师，请问下对于插拔网线导致连接断开，websocket的服务端或客户端，可以监听到这个事件吗？是不是还需要程序做一些补偿机制？

作者回复: socket连接是TCP传输层负责维护的，WebSocket是应用层协议，实际不管这个。



澳

2019-06-20

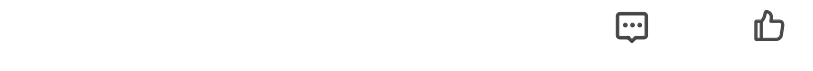
老师，在实际使用中，websocket和stomp over websocket优缺点如何？web在线聊天的语音或图片消息也是通过websocket实现，还是有其他技术？



门窗小二

2019-06-20

通过从session中找到要发送的目标对象的信息！然后进行匹配发送

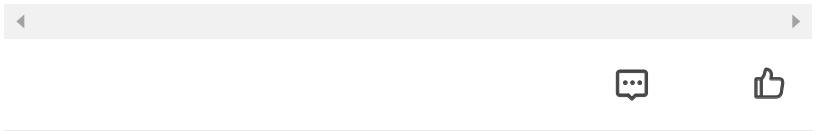


crazypokerk

2019-06-20

请问下老师，WebSocket和Servlet容器中处理请求的Servlet算是同一等级的组件吗？

作者回复: 是的

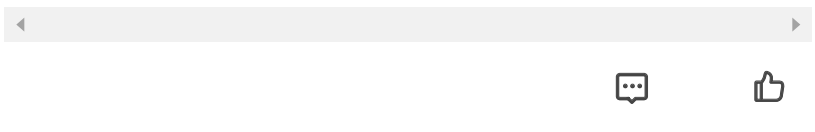


**Liam**

2019-06-20

维护一个全局用户连接字典，建立连接时，将用户连接加入字典，发送消息时，从字典获取特定用户的连接并发送？

作者回复: 对的



**罗乾林**

2019-06-20

向指定用户发送消息：添加登录机制，用户登录时维护用户id和session的关联关系。客户端向指定用户发送消息时指定用户id 发送给server，server通过用户id获取session转发消息



