xNPE Lv2
2018年08月13日 阅读 5108

关注

WebSocket 的故事（二）—— Spring 中如何利用 STOMP 快速构建 WebSocket 广播式消息模式

概述

本文是WebSocket的故事系列第二篇，WebSocket的故事系列计划分五篇，旨在由浅入深的介绍WebSocket以及在Springboot中如何快速构建和使用WebSocket提供的能力。本系列计划包含如下几篇文章：

第一篇，什么是WebSocket以及它的用途
第二篇，Spring中如何利用STOMP快速构建WebSocket广播式消息模式
第三篇，Springboot中，如何利用WebSocket和STOMP快速构建点对点的消息模式(1)
第四篇，Springboot中，如何利用WebSocket和STOMP快速构建点对点的消息模式(2)
第五篇，Springboot中，实现网页聊天室之自定义WebSocket消息代理
第六篇，Springboot中，实现更灵活的WebSocket

本篇的主线

承接上文对WebSocket的介绍，由WebSocket的发送接收信息谈起，对STOMP协议做大致介绍，最后，通过Springboot和JS，实际编写一个WebSocket例子，实现广播式消息发送。

本篇适合的读者

想要了解STOMP协议，以及如何使用Springboot搭建WebSocket服务的同学。

承上启下

WebSocket的开发者，都需要自己在服务端和客户端定义一套规则，来传输信息。那么，有没有已经造好的轮子呢？答案肯定是有的。这就是**STOMP**。

STOMP(Simple Text Oriented Messaging Protocol)简介

STOMP是一个用于C/S之间进行异步消息传输的简单文本协议, 全称是Simple Text Oriented Messaging Protocol。

> STOMP官方网站

其实STOMP协议并不是为WS所设计的, 它其实是消息队列的一种协议, 和AMQP,JMS是平级的。 只不过由于它的简单性恰巧可以用于定义WS的消息体格式。 目前很多服务端消息队列都已经支持了STOMP, 比如RabbitMQ, Apache ActiveMQ等。很多语言也都有STOMP协议的客户端解析库，像JAVA的Gozirra，C的libstomp，Python的pyactivemq，JavaScript的stomp.js等等。

STOMP协议

STOMP是一种基于帧的协议，一帧由一个命令，一组可选的Header和一个可选的Body组成。STOMP是基于Text的，但也允许传输二进制数据。 它的默认编码是UTF-8，但它的消息体也支持其他编码方式，比如压缩编码。

STOMP服务端

STOMP服务端被设计为客户端可以向其发送消息的一组目标地址。STOMP协议并没有规定目标地址的格式，它由使用协议的应用自己来定义。 例如/topic/a，/queue/a，queue-a对于STOMP协议来说都是正确的。应用可以自己规定不同的格式以此来表明不同格式代表的含义。比如应用自己可以定义以/topic打头的为发布订阅模式，消息会被所有消费者客户端收到，以/user开头的为点对点模式，只会被一个消费者客户端收到。

STOMP客户端

对于STOMP协议来说, 客户端会扮演下列两种角色的任意一种：

- 作为生产者，通过SEND帧发送消息到指定的地址

实际上，WebSocket结合STOMP相当于构建了一个消息分发队列，客户端可以在上述两个角色间转换，订阅机制保证了一个客户端消息可以通过服务器广播到多个其他客户端，作为生产者，又可以通过服务器来发送点对点消息。

STOMP帧结构

```
COMMAND
header1:value1
header2:value2

Body^@
```

^@表示行结束符

一个STOMP帧由三部分组成:命令，Header(头信息)，Body（消息体）

- 命令使用UTF-8编码格式，命令有SEND、SUBSCRIBE、MESSAGE、CONNECT、CONNECTED等。
- Header也使用UTF-8编码格式，它类似HTTP的Header，有content-length,content-type等。
- Body可以是二进制也可以是文本。注意Body与Header间通过一个空行（EOL）来分隔。

来看一个实际的帧例子：

```
SEND
destination:/broker/roomId/1
content-length:57

{“type":"ENTER","content":"o7jD64gNifq-wq-C13Q5CRisJx5E"}
```

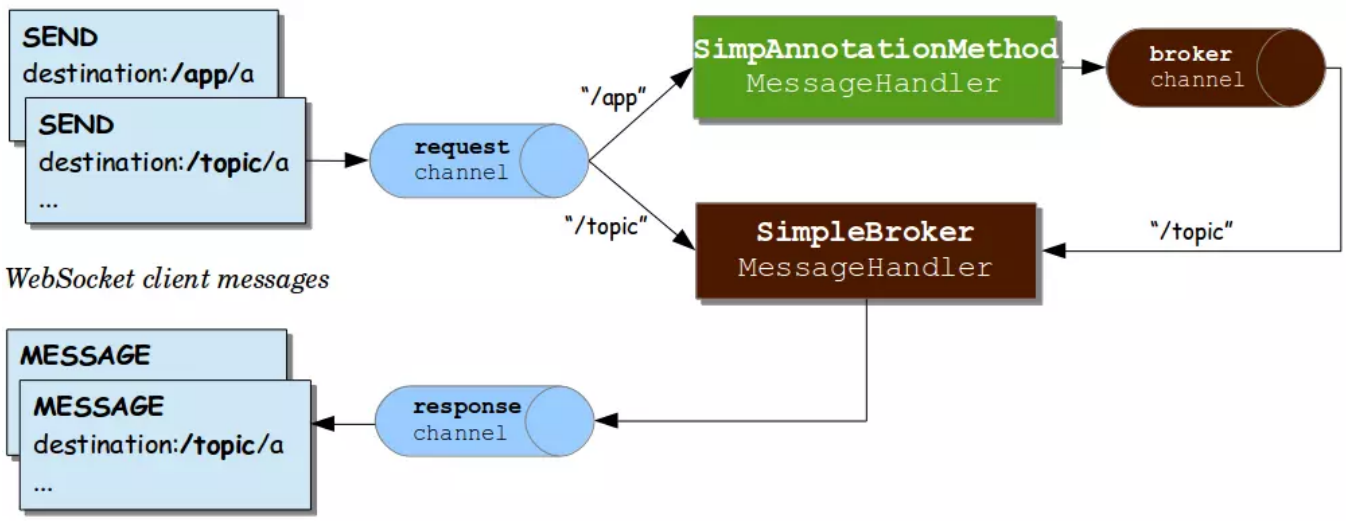
- 第1行：表明此帧为SEND帧，是COMMAND字段。
- 第2行：Header字段，消息要发送的目的地址，是相对地址。
- 第3行：Header字段，消息体字符长度。
- 第4行：空行，间隔Header与Body。
- 第5行：消息体，为自定义的JSON结构。

更多STOMP协议的细节，如果大家感兴趣，可以参考上述的官方网页，有更多详细的帧结构介绍

面，我们将主要介绍用Springboot和JS实现后端和前端，构建一个WebSocket的小型应用场景。

Spring中的WebSocket架构

架构图



图中各个组件介绍：

- **生产者型客户端（左上组件）**：发送SEND命令到某个目的地址(destination)的客户端。
- **消费者型客户端（左下组件）**：订阅某个目的地址(destination), 并接收此目的地址所推送过来的消息的客户端。
- **request channel**: 一组用来接收生产者型客户端所推送过来的消息的线程池。
- **response channel**: 一组用来推送消息给消费者型客户端的线程池。
- **broker**: 消息队列管理者，也可以成为消息代理。它有自己的地址（例如"/topic"），客户端可以向其发送订阅指令，它会记录哪些订阅了这个目的地址(destination)。
- **应用目的地址(图中的"/app")**: 发送到这类目的地址的消息在到达broker之前，会先路由到由应用写的某个方法。相当于对进入broker的消息进行一次拦截，目的是针对消息做一些业务处理。
- **非应用目的地址(图中的"/topic"**，也是消息代理地址): 发送到这类目的地址的消息会直接转到broker。不会被应用拦截。
- **SimpAnnotatonMethod**: 发送到应用目的地址的消息在到达broker之前, 先路由到的方法. 这部分代码是由应用控制的。

消息从生产者发出到消费者消费的流转流程

首先，生产者通过发送一条SEND命令消息到某个目的地址(destination)，服务端request channel接受到这条SEND命令消息，如果目的地址是应用目的地址则转到相应的由应用自己写的业务方法做处（对应图中的SimpAnnotationMethod），再转到broker(SimpleBroker)。如果目的地址是非应用目的

Spring运用WebSocket实现简单的广播消息

场景描述

我们来实现一个简单聊天室的第一步，每当有用户加入聊天室时，该用户向服务器发送加入聊天室的消息，服务器向当前聊天室内的所有用户发送欢迎语。

创建Message-handling Controller

在Spring中，STOMP消息会被路由到以**Controller**注解标识的类中。即我们需要定义一个控制器类，并使用**Controller**注解来标识它，然后在其中实现具体的消息处理方法，我们创建一个名为**GreetingController**的类：

```
package com.xnpe.club.wbs.controller;

import com.xnpe.club.wbs.data.Greeting;
import com.xnpe.club.wbs.data.HelloMessage;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;
import org.springframework.web.util.HtmlUtils;

@Controller //使用Controller注解来标识这是一个控制器类
public class GreetingController {

    @MessageMapping("/hello") //使用MessageMapping注解来标识所有发送到“/hello”这个destination的消
    @SendTo("/topic/greetings") //使用SendTo注解来标识这个方法返回的结果，都会被发送到它指定的destin
    //传入的参数HelloMessage为客户端发送过来的消息，是自动绑定的。
    public Greeting greeting(HelloMessage message) throws Exception {
        Thread.sleep(1000); // 模拟处理延时
        return new Greeting("Hello, " + HtmlUtils.htmlEscape(message.getName()) + "!"); //相
    }

}
```

整体下来，greeting()方法的作用是，处理所有发到/hello这个destination的信息，并将处理的结果发送到所有订阅了/topic/greetings这个destination的客户端。其中模拟的延时，其本质是为了演示在WebSocket中，我们无需考虑超时这样的问题，即上一篇文章提到的，客户端与服务端连接建立后，服务端可以根据实际场景，在“任何有需要”的时候“推送”消息到客户端，直到连接释放。

刚才我们已经创建了消息处理控制器，也就是我们的业务处理逻辑。现在我们要为Spring配置WebSocket和STOMP消息设置。 创建一个名为WebSocketController的类：

```
package com.xnpe.club.wbs.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;

@Configuration //使用Configuration注解标识这是一个Springboot的配置类.
@EnableWebSocketMessageBroker //使用此注解来标识使能WebSocket的broker.即使用broker来处理消息.
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    //实现WebSocketMessageBrokerConfigurer中的此方法，配置消息代理（broker）
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic"); //启用SimpleBroker，使得订阅到此"topic"前缀的客户端
        config.setApplicationDestinationPrefixes("/app"); //将"app"前缀绑定到MessageMapping注
    }

    @Override
    //用来注册Endpoint，“/gs-guide-websocket”即为客户端尝试建立连接的地址。
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/gs-guide-websocket").withSockJS();
    }

}
```

配置主要包含两部分内容，一个是消息代理，另一个是Endpoint，消息代理指定了客户端订阅地址，以及发送消息的路由地址；Endpoint指定了客户端建立连接时的请求地址。

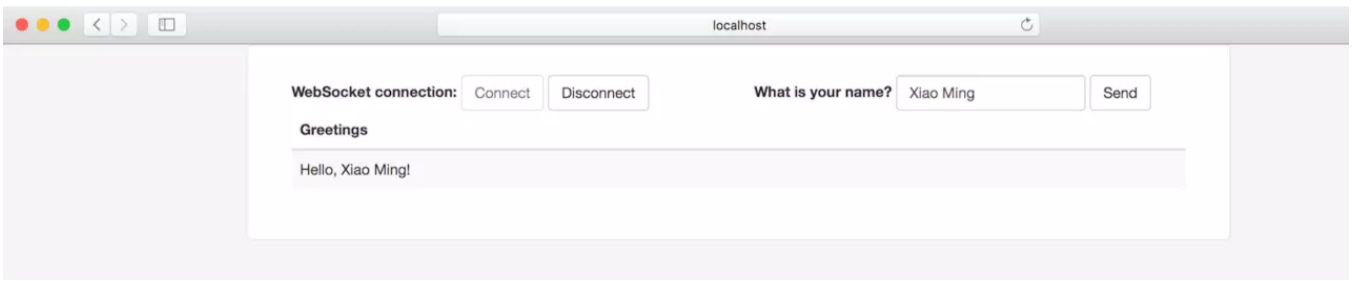
至此，服务端的配置工作就完成了，非常简单。现在，让我们实现一个前端页面，来验证服务的工作情况。

创建前端实现页面

针对STOMP，前端我们采用JavaScript的stomp的客户端实现stomp.js以及WebSocket的实现SockJS。此处只展示核心代码。

```
    var socket = new SockJS('/gs-guide-websocket');
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function (frame) {
        //连接成功后的回调方法
        setConnected(true);
        console.log('Connected: ' + frame);
        //订阅/topic/greetings地址，当服务端向此地址发送消息时，客户端即可收到。
        stompClient.subscribe('/topic/greetings', function (greeting) {
            //收到消息时的回调方法，展示欢迎信息。
            showGreeting(JSON.parse(greeting.body).content);
        });
    });
}
//断开连接的方法
function disconnect() {
    if (stompClient !== null) {
        stompClient.disconnect();
    }
    setConnected(false);
    console.log("Disconnected");
}
//将用户输入的名字信息,使用STOMP客户端发送到“/app/hello”地址。它正是我们在GreetingController中定义的g
function sendName() {
    stompClient.send("/app/hello", {}, JSON.stringify({'name': $("#name").val()}));
}
```

演示



点击“Connect”按钮后，如果连接成功，Connect按钮会置灰；输入名字后点击Send，服务端会返回欢迎语。

参考代码

总结

至此，我们实现了一个最简单的使用Spring，基于STOMP的WebSocket例子。下一篇我们会基于这个例子，继续完善聊天室功能，实现点对点的通信功能。即两个用户如何点对点的聊天，敬请期待。

小铭出品，必属精品

欢迎关注xNPE技术论坛，更多原创干货每日推送。

关注下面的标签，发现更多相似文章

JavaScript 后端 Spring WebSocket

xNPE Lv2
非典型程序员 @ 自由开发者
获得点赞 452 · 获得阅读 23,327

关注

评论

输入评论...

消火零回复，感谢对官方例子的注释，终于解惑了

4月前 回复

相关推荐

专栏 · 幻魂 · 4分钟前 · React.js / JavaScript

应战Vue3 setup，Concent携手React出招了！

专栏 · 叁公子KCN · 12分钟前 · 后端 / Python

图像搜索：给你爬的美女图建一个搜索引擎

专栏 · LinDaiDai_霖呆呆 · 5小时前 · JavaScript

JavaScript进阶-内存空间详解(双十一过后的一更)

4

专栏 · 江五渣 · 12小时前 · Go / 后端

聊聊 Go 语言中的字符表示与字符串遍历

7 7

专栏 · 故事胶片 · 3天前 · JavaScript / 前端

前端Vue中常用rules校验规则

506 39

专栏 · 徐小夕_Lab实验室 · 1天前 · JavaScript / 前端

《前端实战总结》之使用pace.js为你的网站添加加载进度条

98 6

热 · 专栏 · 神三元 · 13天前 · JavaScript

(建议精读)原生JS灵魂之问(中)，检验自己是否真的熟悉JavaScript?

1314 151

专栏 · 刘小夕 · 12天前 · JavaScript / 前端框架

9个项目助你在2020年成为前端大神！

1242 76