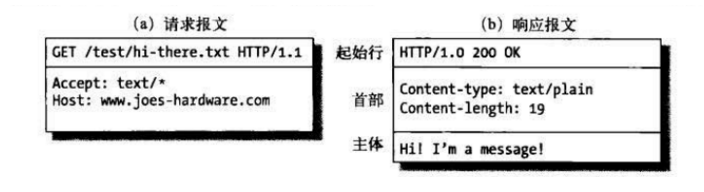
<http://blog.csdn.net/yingxiake/article/details/51213060>

版权声明：本文为博主原创文章，未经博主允许不得转载。

websocket只定义了文字和字节俩种形式的消息格式，没有像http协议那样子有那么丰富的协议规范，我们看看http的协议格式



websocket之所以没有自己定义那么多的协议格式，是希望有框架自己来实现定义这些格式，我们称之为websocket的子协议，sub-protocol。

STOMP，Streaming Text Orientated Message Protocol，是流文本定向消息协议，是一种为MOM(Message Oriented Middleware，面向消息的中间件)设计的简单文本协议。STOMP的消息格式如下：

COMMAND

header1:value1

header2:value2

Body^@

例如发送消息

SEND

destination:/queue/trade

content-type:application/json

content-length:44

{"action":"BUY","ticker":"MMM","shares",44}^@

再看看订阅消息

SUBSCRIBE

id:sub-1

destination:/topic/price.stock.\*

^@

然后服务器进行广播消息

MESSAGE

message-id:nxahklf6-1

subscription:sub-1

destination:/topic/price.stock.MMM

{"ticker":"MMM","price":129.45}^@

详细了解可以看看官网<http://stomp.github.io/stomp-specification-1.1.html>

[**spring**](http://lib.csdn.net/base/javaee) websocket利用STOMP作为websocket的子协议，原因是stomp可以提供一种类似springmvc的编码方式，可以利用注解进行接收和发送消息以及和springmvc进行无缝的结合。

这样子，我们利用注解来接受和广播websocket信息

@Controller

@RequestMapping("/webSocket")

@MessageMapping("foo")

public class WebSocketController {

@MessageMapping("handle")

@SendTo("/topic/greetings")

public String handle(Task task ,String name) {

//...

return "handle2";

}

}

@MessageMapping即可以用来类级别上也可以用在方法级别上，类似springmvc，但是为不同的是，springmvc的路径是类路径/方法路径，中间是“/”，例如/webSocket/xxxx,而websocket是用“.”来分开路径的，类路径.方法路径，例如foo.handle来发送消息到特定路径

@SendTo可以把消息广播到路径上去，例如上面可以把消息广播到”/topic/greetings”,如果客户端在这个路径订阅消息，则可以接收到消息

接下来需要注册下基于stomp子协议的websocket到spring中

@Configuration

@EnableWebSocketMessageBroker

public class WebSocketMessageBrokerConfig implements WebSocketMessageBrokerConfigurer {

@Override

public void registerStompEndpoints(StompEndpointRegistry registry) {

//portfolio-stomp就是websocket的端点，客户端需要注册这个端点进行链接，withSockJS允许客户端利用sockjs进行浏览器兼容性处理

registry.addEndpoint("/portfolio-stomp").withSockJS();

}

@Override

public void configureMessageBroker(MessageBrokerRegistry registry) {

registry.enableSimpleBroker("/topic"); //设置服务器广播消息的基础路径

registry.setApplicationDestinationPrefixes("/app"); //设置客户端订阅消息的基础路径

registry.setPathMatcher(new AntPathMatcher(".")); //可以已“.”来分割路径，看看类级别的@messageMapping和方法级别的@messageMapping

}

@Override

public boolean configureMessageConverters(List<MessageConverter> messageConverters) {

return true;

}

@Override

public void configureWebSocketTransport(WebSocketTransportRegistration registry) {

// TODO Auto-generated method stub

registry.addDecoratorFactory(new MyWebSocketHandlerDecoratorFactory());

}

@Override

public void configureClientInboundChannel(ChannelRegistration registration) {

registration.setInterceptors(new MyChannelInterceptor());

}

@Override

public void configureClientOutboundChannel(ChannelRegistration registration) {

// TODO Auto-generated method stub

}

@Override

public void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolvers) {

// TODO Auto-generated method stub

}

@Override

public void addReturnValueHandlers(List<HandlerMethodReturnValueHandler> returnValueHandlers) {

// TODO Auto-generated method stub

}

}

前端需要用到sockjs.[**js**](http://lib.csdn.net/base/javascript)和stomp.js

var socket = new SockJS('/whats/portfolio-stomp');

/\*\*

\* 建立成功的回调函数

\*/

socket.onopen = function() {

console.log('open');

};

/\*\*

\* 服务器有消息返回的回调函数

\*/

socket.onmessage = function(e) {

console.log('message', e.data);

};

/\*\*

\* websocket链接关闭的回调函数

\*/

socket.onclose = function() {

console.log('close');

};

var stompClient = Stomp.over(socket);

stompClient.connect({}, function(frame) {

stompClient.subscribe('/topic/greetings', function(data) { //订阅消息

$("#ret").text(data);

});

});

document.getElementById("ws").onclick = function() {

stompClient.send("/app/foo.handle3", {}, JSON.stringify({

name : "nane",

taskName : "taskName",

taskDetail : "taskDetail"

}));

}

到这里就可以就服务器就可以接收和广播消息了，而客户端就可以发送和订阅消息了。

类中的handle的方法参数绑定接收和格式化的处理类似springmvc中方法参数的处理

而方法参数的返回值会被MessageConverter进行转化封装然后发送给广播出去，类似springmvc的方法中带@responsebody注解时候方法返回值会被httpMessageConverter进行转化一样。

spring websocket的方法参数中还允许有其他一些参数如下

Principal principal ,存放用户的登录验证信息   
Message message，最基础的消息体，里面方有header和payload等信息   
@Payload 消息体内容   
@Header(“..”) 某个头部key的值   
@Headers, 所有头部key的map集合   
MessageHeaders , SimpMessageHeaderAccessor, MessageHeaderAccessor ,StompHeaderAccessor 消息头信息   
@DestinationVariable 类似springmvc中的@PathVariable

在spring websocket中也可以类似springmvc中把变量绑定在路径上，例如

@MessageMapping("bar.{baz}")

@SendTo("/topic/greetings")

public String handle1(@DestinationVariable String baz) {

return baz;

}

这就是注解大概的使用方式，如何已注解方式来进行精准推送和全部推送呢？？