# 书的资源

## <http://www.broadview.com.cn/book/472>

# 1. 服务器端推送技术

## 1.Controller ,输出类型为produces="text/event-stream") ，只是服务器端对sse的支持，每5秒向浏览器随机推送消息

@RequestMapping(value=**"/push"**,produces=**"text/event-stream"**) *//1***public** @ResponseBody String push(){  
 Random r = **new** Random();   
 **try** {  
 Thread.*sleep*(5000);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }   
 **return "data:Testing 1,2,3"** + r.nextInt() +**"\n\n"**;  
}

## 2. html端 EventSource，目前只有新式的浏览器才支持

<**script type="text/javascript"**>  
  
  
 **if** (!!**window**.EventSource) { *//1* **var *source*** = **new** EventSource(**'push'**);   
 **s**=**''**;  
 ***source***.addEventListener(**'message'**, **function**(e) {*//2* **s**+=e.data+**"<br/>"**;  
 **$**(**"#msgFrompPush"**).html(**s**);  
   
 });  
  
 ***source***.addEventListener(**'open'**, **function**(e) {  
 **console**.log(**"连接打开."**);  
 }, **false**);  
  
 ***source***.addEventListener(**'error'**, **function**(e) {  
 **if** (e.**readyState** == EventSource.**CLOSED**) {  
 **console**.log(**"连接关闭"**);  
 } **else** {  
 **console**.log(e.**readyState**);   
 }  
 }, **false**);  
 } **else** {  
 **console**.log(**"你的浏览器不支持SSE"**);  
 }   
</**script**>  
</**body**>  
</**html**>

# 2. jackson 对对象和json的转化一定需要此空构造

# 3. 拦截器

## 1.拦截器1

**public class** MyInterceptor1 **implements** HandlerInterceptor {  
  
 @Override  
 **public boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)  
 **throws** Exception {  
 System.***out***.println(**">>>MyInterceptor1>>>>>>>在请求处理之前进行调用（Controller方法调用之前）"**);  
  
 **return true**;*// 只有返回true才会继续向下执行，返回false取消当前请求* }  
  
 @Override  
 **public void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,  
 ModelAndView modelAndView) **throws** Exception {  
 System.***out***.println(**">>>MyInterceptor1>>>>>>>请求处理之后进行调用，但是在视图被渲染之前（Controller方法调用之后）"**);  
 }  
  
 @Override  
 **public void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)  
 **throws** Exception {  
 System.***out***.println(**">>>MyInterceptor1>>>>>>>在整个请求结束之后被调用，也就是在DispatcherServlet 渲染了对应的视图之后执行（主要是用于进行资源清理工作）"**);  
 }

## 2.拦截器2

**public class** MyInterceptor2 **implements** HandlerInterceptor {  
  
 @Override  
 **public boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)  
 **throws** Exception {  
 System.***out***.println(**">>>MyInterceptor2>>>>>>>在请求处理之前进行调用（Controller方法调用之前）"**);  
  
 **return true**;*// 只有返回true才会继续向下执行，返回false取消当前请求* }  
  
 @Override  
 **public void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,  
 ModelAndView modelAndView) **throws** Exception {  
 System.***out***.println(**">>>MyInterceptor2>>>>>>>请求处理之后进行调用，但是在视图被渲染之前（Controller方法调用之后）"**);  
 }  
  
 @Override  
 **public void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)  
 **throws** Exception {  
 System.***out***.println(**">>>MyInterceptor2>>>>>>>在整个请求结束之后被调用，也就是在DispatcherServlet 渲染了对应的视图之后执行（主要是用于进行资源清理工作）"**);  
 }  
  
}

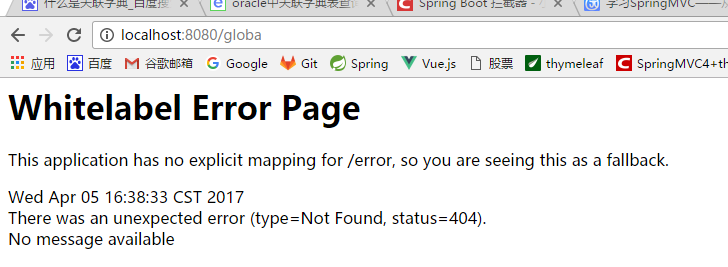
## 3.添加配置文件类，设置拦截器

### 1.配置文件类使用configuration注解，这样启动的时候，会自动加载这个配置文件

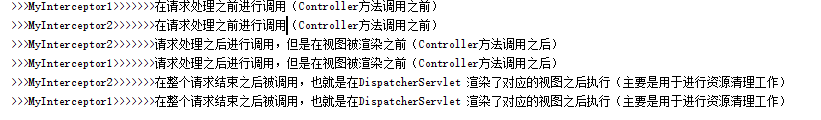
### 2.配置文件需要spring mvc支持，所以需要继承WebMvcConfigurerAdapter

@Configuration  
**public class** MyWebAppConfigurer **extends** WebMvcConfigurerAdapter {  
  
 *//添加拦截器* @Override  
 **public void** addInterceptors(InterceptorRegistry registry) {  
 *// 多个拦截器组成一个拦截器链  
 // addPathPatterns 用于添加拦截规则  
 // excludePathPatterns 用户排除拦截* registry.addInterceptor(**new** MyInterceptor1()).addPathPatterns(**"/\*\*"**);  
 registry.addInterceptor(**new** MyInterceptor2()).addPathPatterns(**"/\*\*"**);  
 **super**.addInterceptors(registry);  
 }  
  
  
  
}

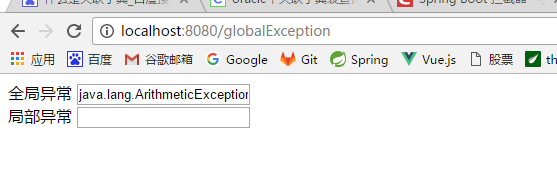
## 4.测试1 地址随意



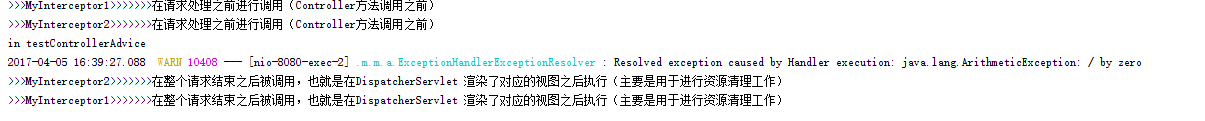
### 控制台打印结果



## 4测试2.地址路径controller发送异常



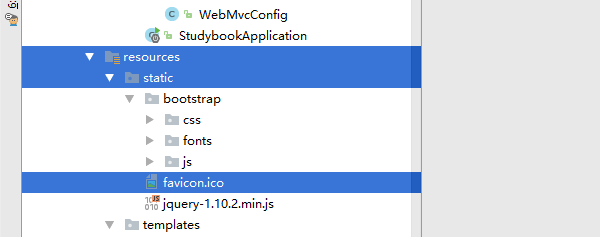
### 控制台打印结果



## 5.解释两种测试区别

第二种controller中发送了异常，虽然controller调用了，但是并没有走完controller，所以这里的信息不打印postHandle 方法

# 4.设置自己的浏览器图标favicon



# 5.WebSocket广播式（顾名思义：服务器有消息时，会将消息发送给所有连接endpoint的用户）

## 1.编写配置文件WebSocketConfig继承AbstractWebSocketMessageBrokerConfigurer重写它的方法来配置websocket

*/\*\** **\* 1 通过 @EnableWebSocketMessageBroker  
 \* 开启使用stomp协议来传输 基于代理的 message broker,  
 \* 这个时候控制器支持使用MessageMapping** *\* \*/*@Configuration  
@EnableWebSocketMessageBroker *//1***public class** WebSocketConfig **extends** AbstractWebSocketMessageBrokerConfigurer {

## 2. 实现其方法

*/\*\**  \* 注册Stomp的协议的节点，并映射指定的url *\** ***@param registry*** *\*/*@Override  
 **public void** registerStompEndpoints(StompEndpointRegistry registry) {  
 //注册一个stomp的节点，并指定使用SockJS协议，将来用于浏览器用户与服务器连接registry.addEndpoint(**"/endpointWisely"**).withSockJS();  
 }  
  
 */\*\**  \* 配置信息代理 *\** ***@param registry*** *\*/* @Override  
 **public void** configureMessageBroker(MessageBrokerRegistry registry) {  
 //广播式应配置一个topic的消息代理，这个讲来用于用户通过浏览器进行订阅registry.enableSimpleBroker(**"/topic"**); *//2* }

## 3. 浏览器给服务器发送消息的类

*/\*\*  
 \* 浏览器向服务器发送的消息，用这个接受  
 \*/***public class** WiselyMessage {  
 **private** String **name**;  
  
 **public** String getName(){  
 **return name**;  
 }  
}

## 4.服务器给浏览器发消息

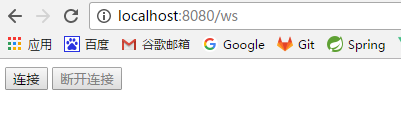
*/\*\*  
 \* 服务器向浏览器发送从此勒的消息  
 \*/***public class** WiselyResponse {  
 **private** String **responseMessage**;  
 **public** WiselyResponse(String responseMessage){  
 **this**.**responseMessage** = responseMessage;  
 }  
 **public** String getResponseMessage(){  
 **return responseMessage**;  
 }  
}

## 5. 添加控制器，实现与用户交互

@Controller  
**public class** WsController {  
 @MessageMapping(**"/welcome"**) //用户发送信息时候的地址  
 @SendTo(**"/topic/getResponse"**) *//服务器有消息时候，会对订阅了这个路径中的浏览器发送消息* **public** WiselyResponse say(WiselyMessage message) **throws** Exception{  
 Thread.*sleep*(3000);  
 **return new** WiselyResponse(**"welcome,"**+message.getName());  
 }  
}

## 6.编写html

<**body onload="***disconnect*()**"**>  
<**noscript**><**h2 style="color**: **#ff0000"**>貌似你的浏览器不支持websocket</**h2**></**noscript**>  
<**div**>  
 <**div**>  
 <**button id="connect" onclick="***connect*();**"**>连接</**button**>  
 <**button id="disconnect" disabled="disabled" onclick="***disconnect*();**"**>断开连接</**button**>  
 </**div**>  
 <**div id="conversationDiv"**>  
 <**label**>输入你的名字</**label**><**input type="text" id="name"** />  
 <**button id="sendName" onclick="***sendName*();**"**>发送</**button**>  
 <**p id="response"**></**p**>  
 </**div**>  
</**div**>



## 1点击连接执行connect方法

### 1.连接SockJS的节点的名称为endpointWisely'

### 2.使用stomp的子协议协议的websocket的客户端

### 3.连接websocket的客户端

### 4. *stompClient*.subscribe('/topic/getResponse', function(respnose) 客户端订阅/topic/getResponse，这个是在上面的sendto中定义的。function(respnose){ 是准备进行显示

**function** *connect*() {  
 **var** socket = **new SockJS**(**'/endpointWisely'**); *//1* ***stompClient*** = **Stomp**.over(socket);  
 ***stompClient***.connect({}, **function**(frame) {  
 *setConnected*(**true**);  
 **console**.log(**'Connected: '** + frame);  
 ***stompClient***.subscribe(**'/topic/getResponse'**, **function**(respnose){ *//2  
 showResponse*(***JSON***.parse(respnose.**body**).responseMessage);  
 });  
 });  
 }

## 2. 点击连接之后进行显示相关的html 标签

**function** *setConnected*(connected) {  
 **document**.getElementById(**'connect'**).**disabled** = connected;  
 **document**.getElementById(**'disconnect'**).**disabled** = !connected;  
 **document**.getElementById(**'conversationDiv'**).**style**.**visibility** = connected ? **'visible'** : **'hidden'**;  
 **$**(**'#response'**).html();  
}

## 3.输入名称，点击发送 ，下面的welcome在控制器中的messageMapping中定义的

**function** *sendName*() {  
 **var** name = **$**(**'#name'**).val();  
 *//3* ***stompClient***.send(**"/welcome"**, {}, ***JSON***.stringify({ **'name'**: name }));  
}

## 4.因为在连接时候我们订阅了消息，所以一旦点击发送就会显示服务发送的下次

**function** *showResponse*(message) {  
 **var** response = **$**(**"#response"**);  
 response.html(message);  
}

# 6.Websocket点对点式(广播式不能解决消息由谁发送，由谁接收)

## 1.使用spring security，这里先不做过多讲解

@Configuration  
@EnableWebSecurity  
**public class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {  
 @Override  
 **protected void** configure(HttpSecurity http) **throws** Exception {  
 http  
 .authorizeRequests()  
 .antMatchers(**"/"**,**"/login"**).permitAll()*//1根路径和/login路径不拦截* .anyRequest().authenticated()  
 .and()  
 .formLogin()  
 .loginPage(**"/login"**) //2登陆页面  
.defaultSuccessUrl(**"/chat"**) //3登陆成功转向该页面.permitAll()  
 .and()  
 .logout()  
 .permitAll();  
 }  
  
 //4 配置两个用户 wyf 和wisely 角色相同@Override  
 **protected void** configure(AuthenticationManagerBuilder auth) **throws** Exception {  
 auth  
 .inMemoryAuthentication()  
 .withUser(**"zyj"**).password(**"zyj"**).roles(**"USER"**)  
 .and()  
 .withUser(**"HealerJean"**).password(**"HealerJean"**).roles(**"USER"**);  
 }  
 *//5忽略静态资源的拦截* @Override  
 **public void** configure(WebSecurity web) **throws** Exception {  
 web.ignoring().antMatchers(**"/resources/static/\*\*"**);  
 }  
  
}

## 2.配置websocket，节点和消息代理

registry.addEndpoint(**"/endpointChat"**).withSockJS();*//1*

registry.enableSimpleBroker(**"/topic"**,**"/queue"**); *//2*

## 3.控制器配置

*/\*\*  
 \* 点对点  
 \*/*@Autowired  
**private** SimpMessagingTemplate **messagingTemplate**;  
@MessageMapping(**"/chat"**)  
**public void** handleChat(Principal principal,String msg){  
 **if** (principal.getName().equals(**"zyj"**)){  
 **messagingTemplate**.convertAndSendToUser(**"HealerJean"**,**"/queue/notifications"**,principal.getName()+**"-send:"**+msg);  
  
 }**else** {  
 **messagingTemplate**.convertAndSendToUser(**"zyj"**,**"/queue/notifications"**,principal.getName()+**"-send:"**+msg);  
 }  
}

## 4.登录页面

<**div th:if="${param.error}"**>  
 无效的账号和密码  
</**div**>  
<**div th:if="${param.logout}"**>  
 你已注销  
</**div**>  
<**form th:action="@{/login}" method="post"**>  
 <**div**><**label**> 账号 : <**input type="text" name="username"**/> </**label**></**div**>  
 <**div**><**label**> 密码: <**input type="password" name="password"**/> </**label**></**div**>  
 <**div**><**input type="submit" value="登陆"**/></**div**>  
</**form**>

## 5.登录成功进入聊天页面

<**form id="wiselyForm"**>  
 <**textarea rows="4" cols="60" name="text"**></**textarea**>  
 <**input type="submit"**/>  
</**form**>  
  
<**script th:inline="javascript"**>  
 **$**(**'#wiselyForm'**).submit(**function**(e){  
 e.preventDefault();  
 **var** text = **$**(**'#wiselyForm'**).find(**'textarea[name="text"]'**).val();  
 *sendSpittle*(text);  
 });  
  
 **var *sock*** = **new SockJS**(**"/endpointChat"**); *//1* **var *stomp*** = **Stomp**.over(***sock***);  
 ***stomp***.connect(**'guest'**, **'guest'**, **function**(frame) {  
 ***stomp***.subscribe(**"/user/queue/notifications"**, *handleNotification*);*//2* });  
  
  
  
 **function** *handleNotification*(message) {  
 **$**(**'#output'**).append(**"<b>Received: "** + message.**body** + **"</b><br/>"**)  
 }  
  
 **function** *sendSpittle*(text) {  
 ***stomp***.send(**"/chat"**, {}, text);*//3* }  
 **$**(**'#stop'**).click(**function**() {***sock***.close()});  
</**script**>  
  
<**div id="output"**></**div**>  
</**body**>

# 6.Bootstrap(开发相应式和移动优先的web应用的最流行的Html，只用一套代码就可以在不同设备上显示想要的视图)

## 1.标准配置 下面这个三个必须是head的头三个标签，其他必须放在后面

<**meta content="text/html;charset=UTF-8"**/>  
<**meta http-equiv="X-UA-Compatible" content="IE=edge"**/>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**/>