Messaging with WebSocket and STOMP

Spring´s low-level WebSocket API

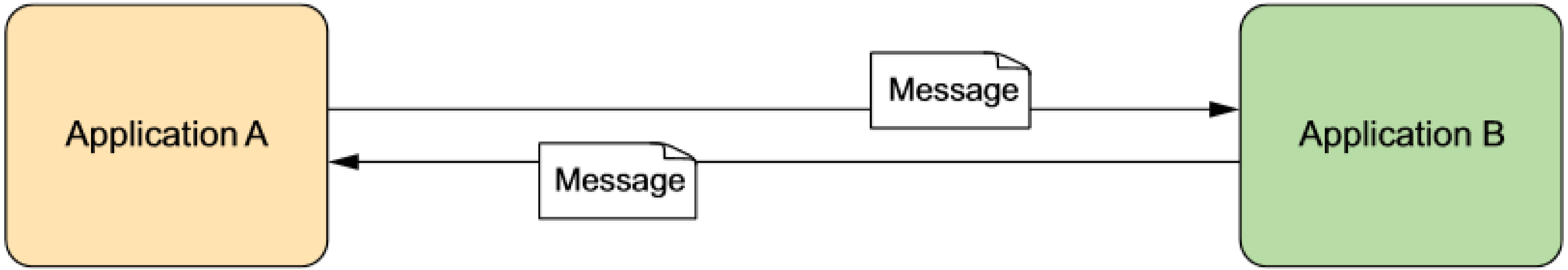
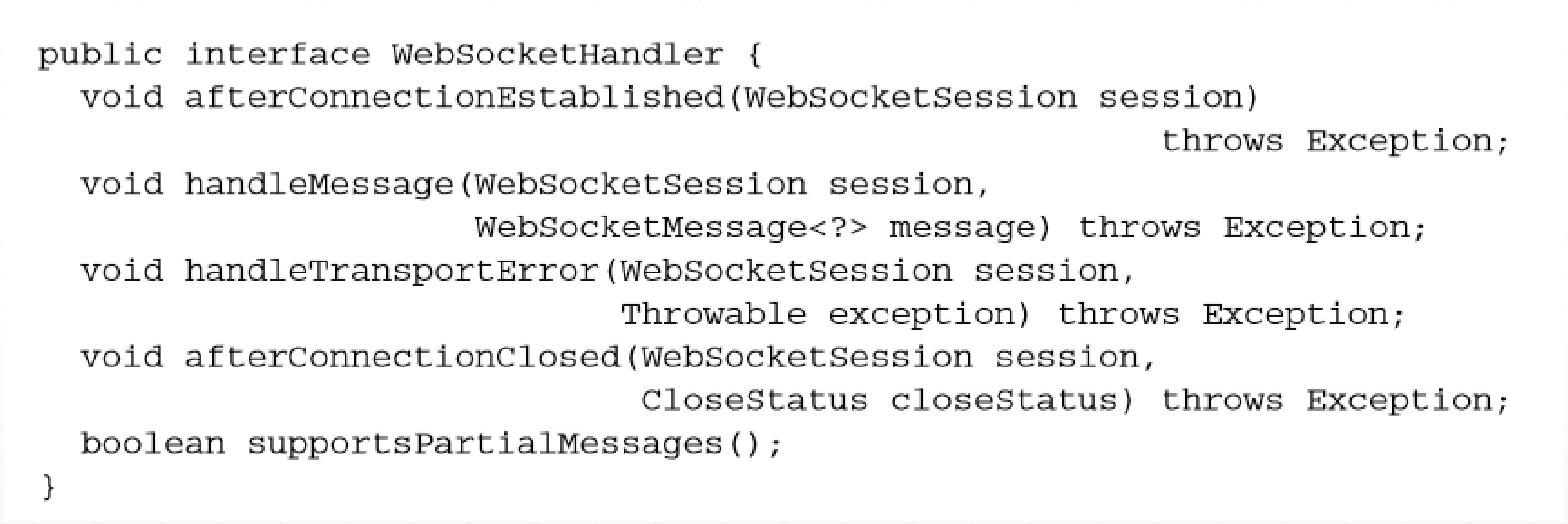


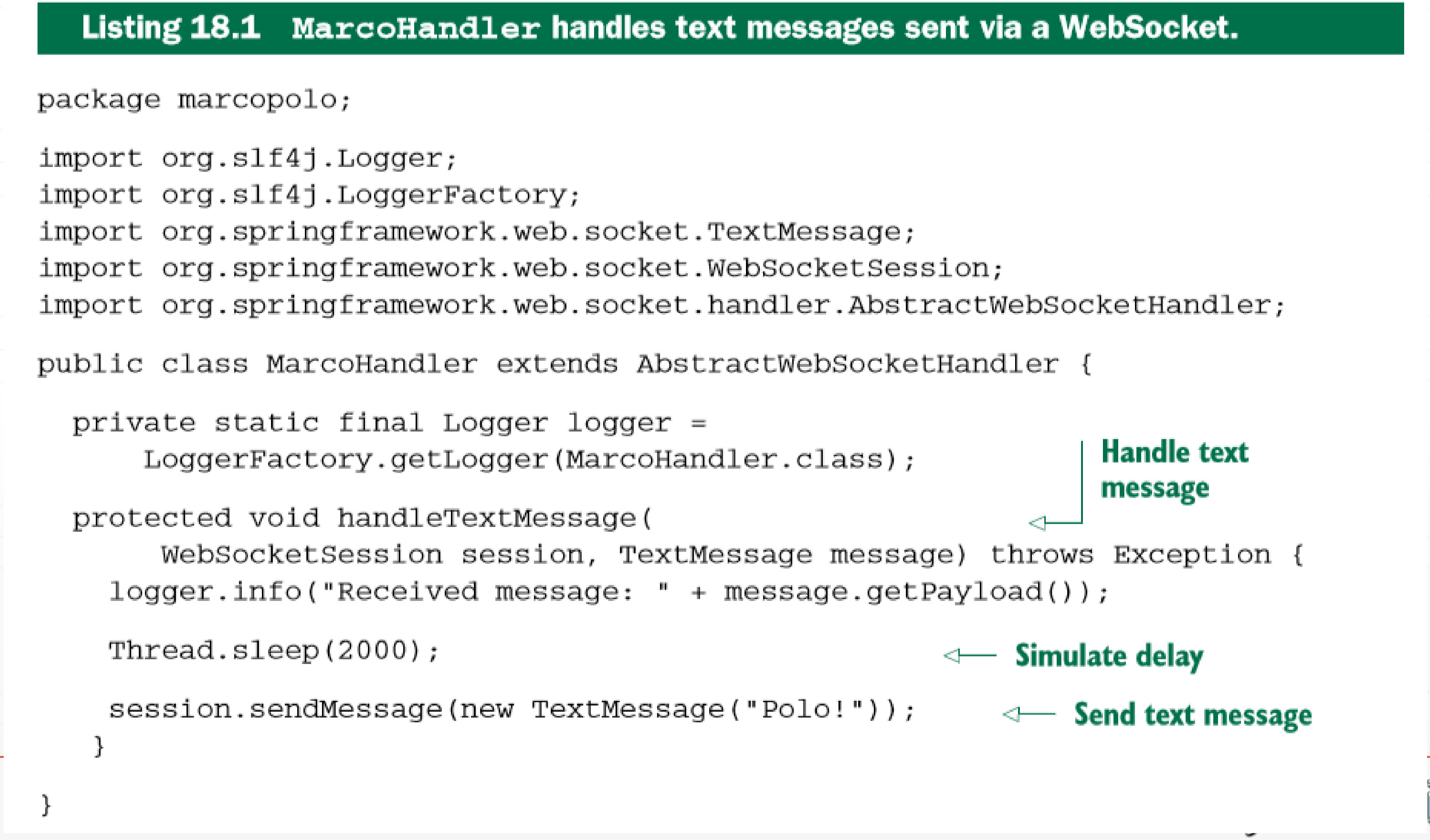
Figure 18.1 A WebSocket is a full-duplex communication channel between two applications.

Writing a simple Marco-Polo Game



```
public interface WebSocketHandler {
  void afterConnectionEstablished(WebSocketSession session)
                                                      throws Exception;
  void handleMessage(WebSocketSession session,
                     WebSocketMessage<?> message) throws Exception;
  void handleTransportError(WebSocketSession session,
                            Throwable exception) throws Exception;
  void afterConnectionClosed(WebSocketSession session,
                             CloseStatus closeStatus) throws Exception;
  boolean supportsPartialMessages();
}
```

Spring´s low-level WebSocket API

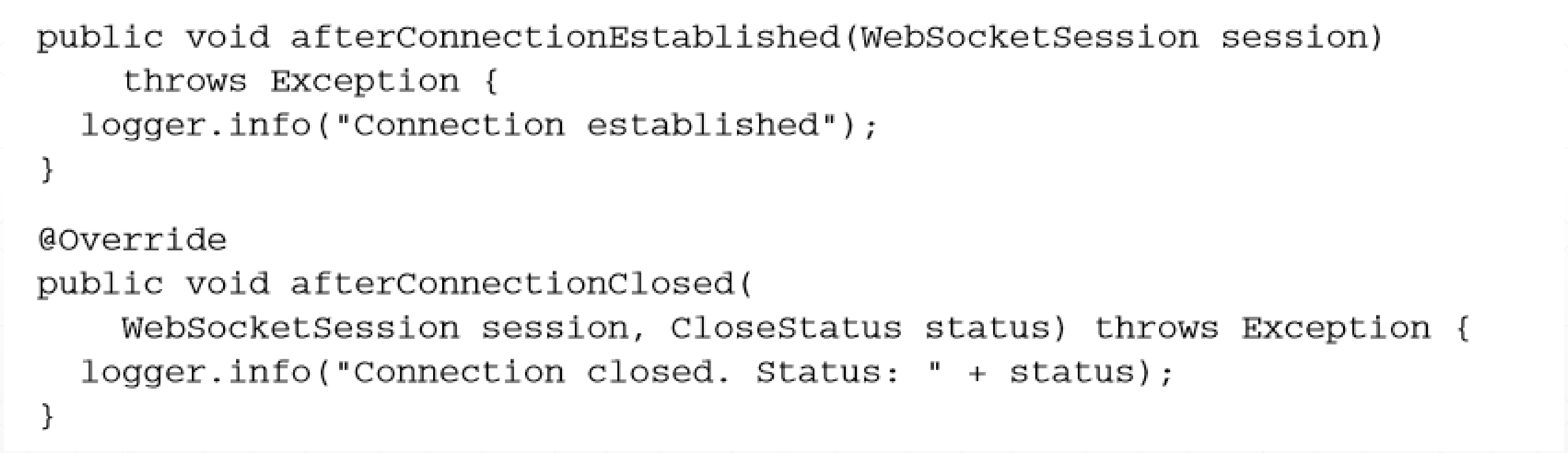


Listing 18.1 `MarcoHandler` **handles text messages sent via a WebSocket.**

```
package marcopolo;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.socket.TextMessage;
import org.springframework.web.socket.WebSocketSession;
import org.springframework.web.socket.handler.AbstractWebSocketHandler;

public class MarcoHandler extends AbstractWebSocketHandler {

  private static final Logger logger =
      LoggerFactory.getLogger(MarcoHandler.class);
                                                                  Handle text
                                                                  message
  protected void handleTextMessage(                         <-
       WebSocketSession session, TextMessage message) throws Exception {
    logger.info("Received message: " + message.getPayload());

    Thread.sleep(2000);                                   <-- Simulate delay

    session.sendMessage(new TextMessage("Polo!"));          <-- Send text message
   }

}
```

Spring´s low-level WebSocket API

- Establishment and closing of connections

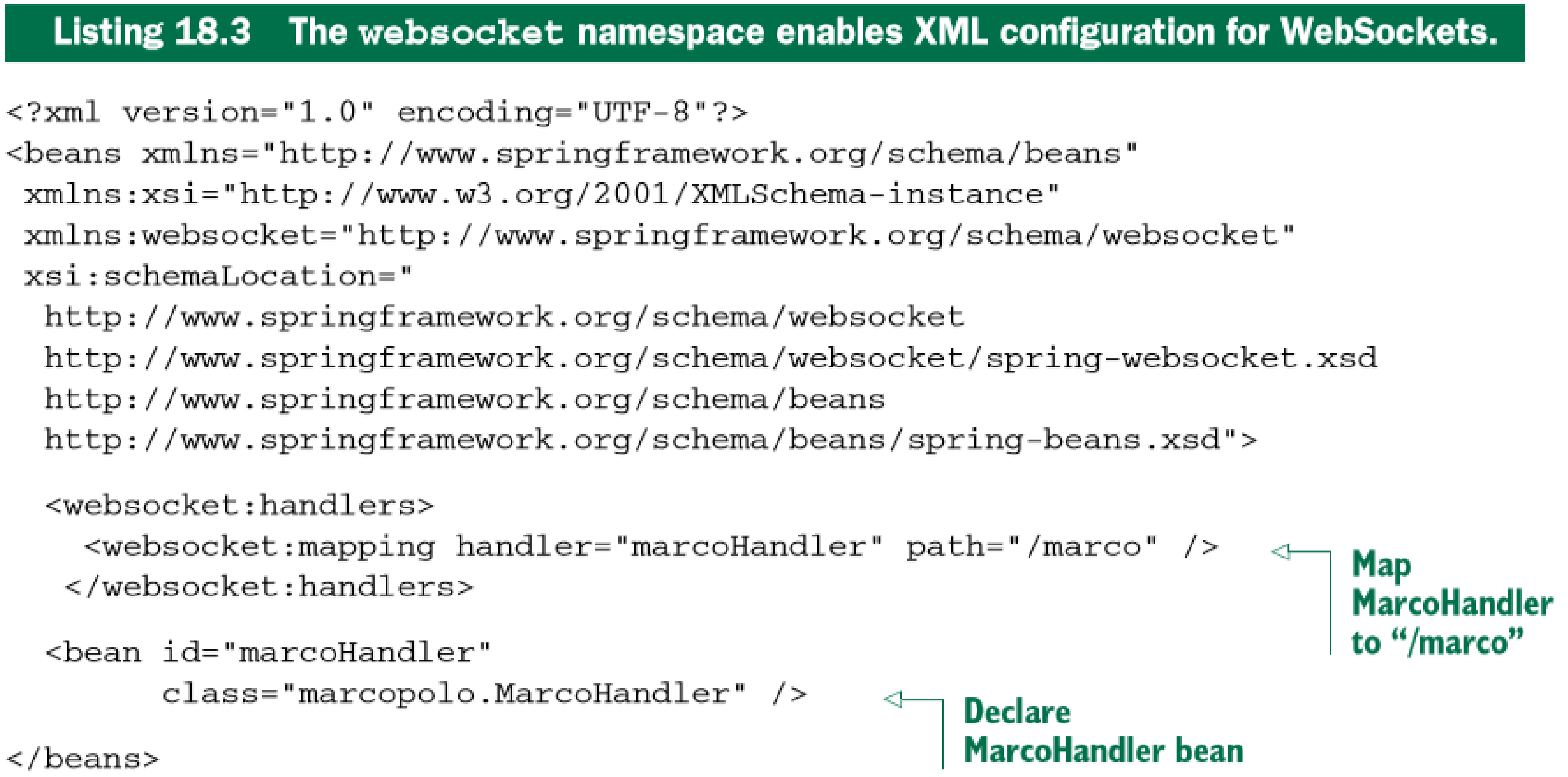


```
public void afterConnectionEstablished(WebSocketSession session)
    throws Exception {
  logger.info("Connection established");
}

@Override
public void afterConnectionClosed(
    WebSocketSession session, CloseStatus status) throws Exception {
  logger.info("Connection closed. Status: " + status);
}
```

Listing 18.2 Enabling WebSocket and mapping a message handler in Java configuration

```
package marcopolo;

import org.springframework.context.annotation.Bean;
import org.springframework.web.socket.config.annotation.
                                              EnableWebSocket;
import org.springframework.web.socket.config.annotation.
                                              WebSocketConfigurer;
import org.springframework.web.socket.config.annotation.
                                              WebSocketHandlerRegistry;

@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

  @Override
  public void registerWebSocketHandlers(
                              WebSocketHandlerRegistry registry) {
    registry.addHandler(marcoHandler(), "/marco");      <-- Map MarcoHandler
   }                                                        to "/marco"

  @Bean
  public MarcoHandler marcoHandler() {      <-- Declare
     return new MarcoHandler();                 MarcoHandler bean
  }

}
```

Spring´s low-level WebSocket API



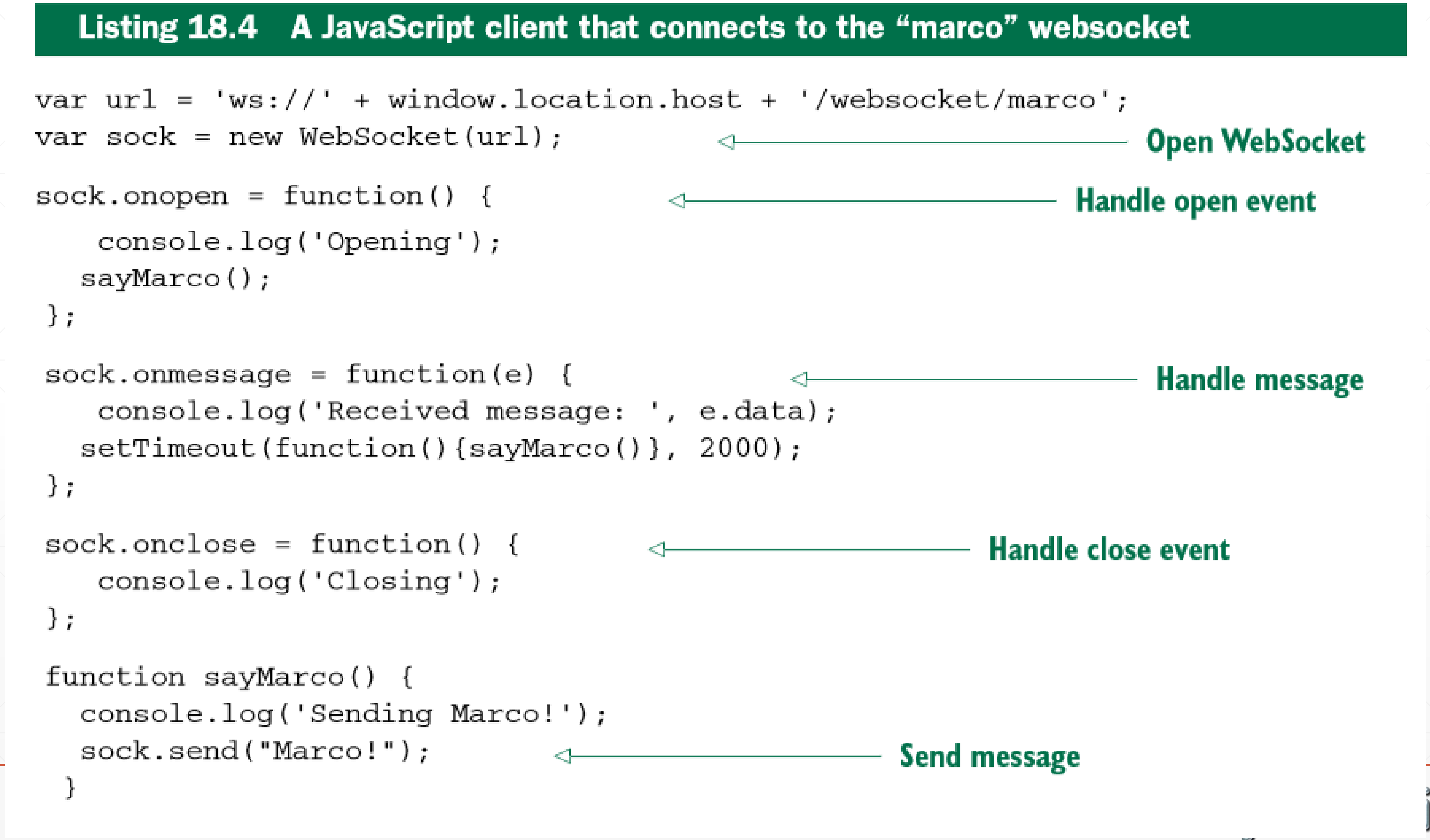
Listing 18.3 The websocket namespace enables XML configuration for WebSockets.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:websocket="http://www.springframework.org/schema/websocket"
 xsi:schemaLocation="
  http://www.springframework.org/schema/websocket
  http://www.springframework.org/schema/websocket/spring-websocket.xsd
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd">

  <websocket:handlers>
    <websocket:mapping handler="marcoHandler" path="/marco" />
   </websocket:handlers>

  <bean id="marcoHandler"
        class="marcopolo.MarcoHandler" />

</beans>
```

Map MarcoHandler to "/marco"

Declare MarcoHandler bean

Simple JavaScript client



Listing 18.4 A JavaScript client that connects to the "marco" websocket

```
var url = 'ws://' + window.location.host + '/websocket/marco';
var sock = new WebSocket(url);                                   Open WebSocket

sock.onopen = function() {                                       Handle open event
    console.log('Opening');
   sayMarco();
 };

 sock.onmessage = function(e) {                                  Handle message
    console.log('Received message: ', e.data);
   setTimeout(function(){sayMarco()}, 2000);
 };

 sock.onclose = function() {                                     Handle close event
    console.log('Closing');
 };

 function sayMarco() {
   console.log('Sending Marco!');
   sock.send("Marco!");                                          Send message
  }
```

Working with STOMP messaging

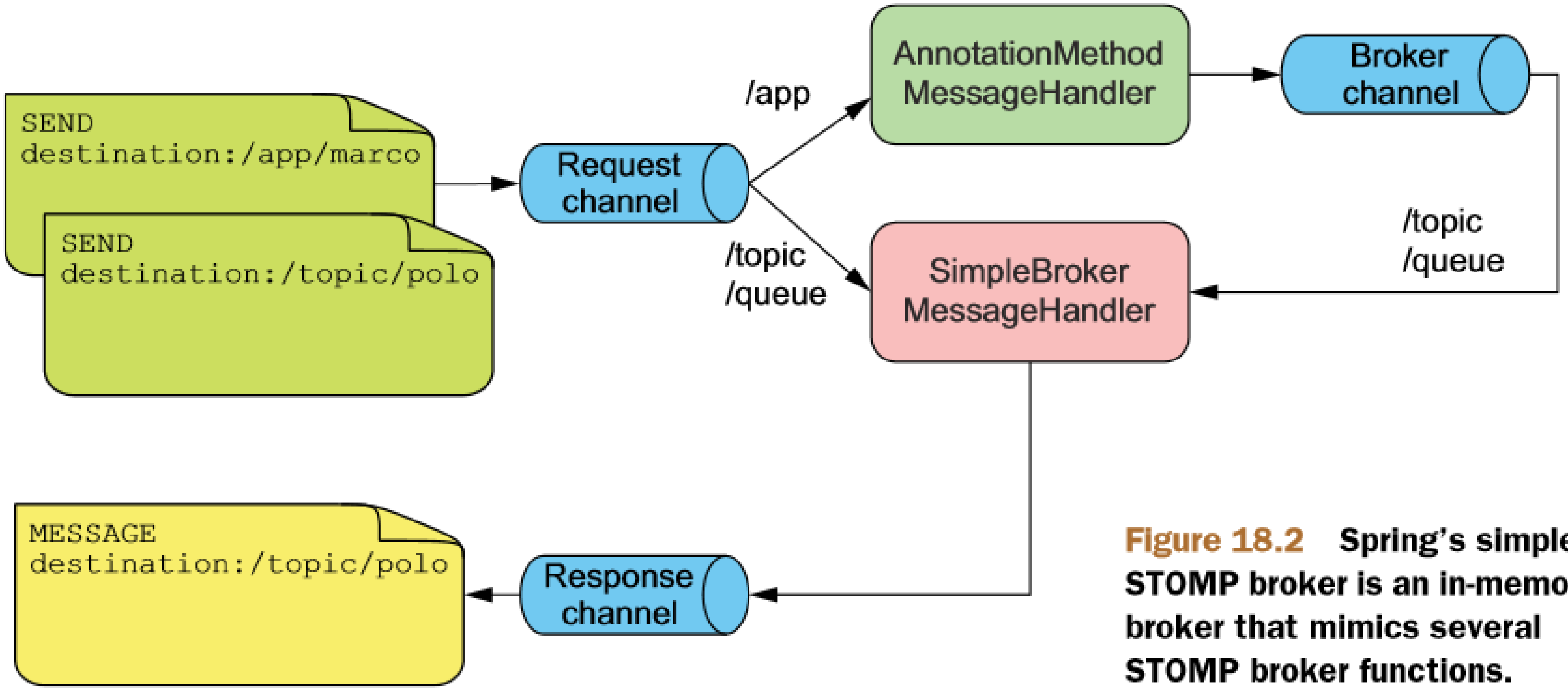


Figure 18.2 **Spring's simple STOMP broker is an in-memory broker that mimics several STOMP broker functions.**

Listing 18.5 @EnableWebSocketMessageBroker enables STOMP over WebSocket.

```
package marcopolo;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.socket.config.annotation.
                              AbstractWebSocketMessageBrokerConfigurer;
import org.springframework.web.socket.config.annotation.
                                          EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.
                                                 StompEndpointRegistry;

@Configuration
@EnableWebSocketMessageBroker                          <-- Enable STOMP messaging
 public class WebSocketStompConfig
       extends AbstractWebSocketMessageBrokerConfigurer {

  @Override
  public void registerStompEndpoints(StompEndpointRegistry registry) {
    registry.addEndpoint("/marcopolo").withSockJS();        <-- Enable SockJS over
   }                                                            /marcopolo

  @Override
  public void configureMessageBroker(MessageBrokerRegistry registry) {
    registry.enableSimpleBroker("/queue", "/topic");
    registry.setApplicationDestinationPrefixes("/app");
  }

}
```

Enabling a STOMP broker relay

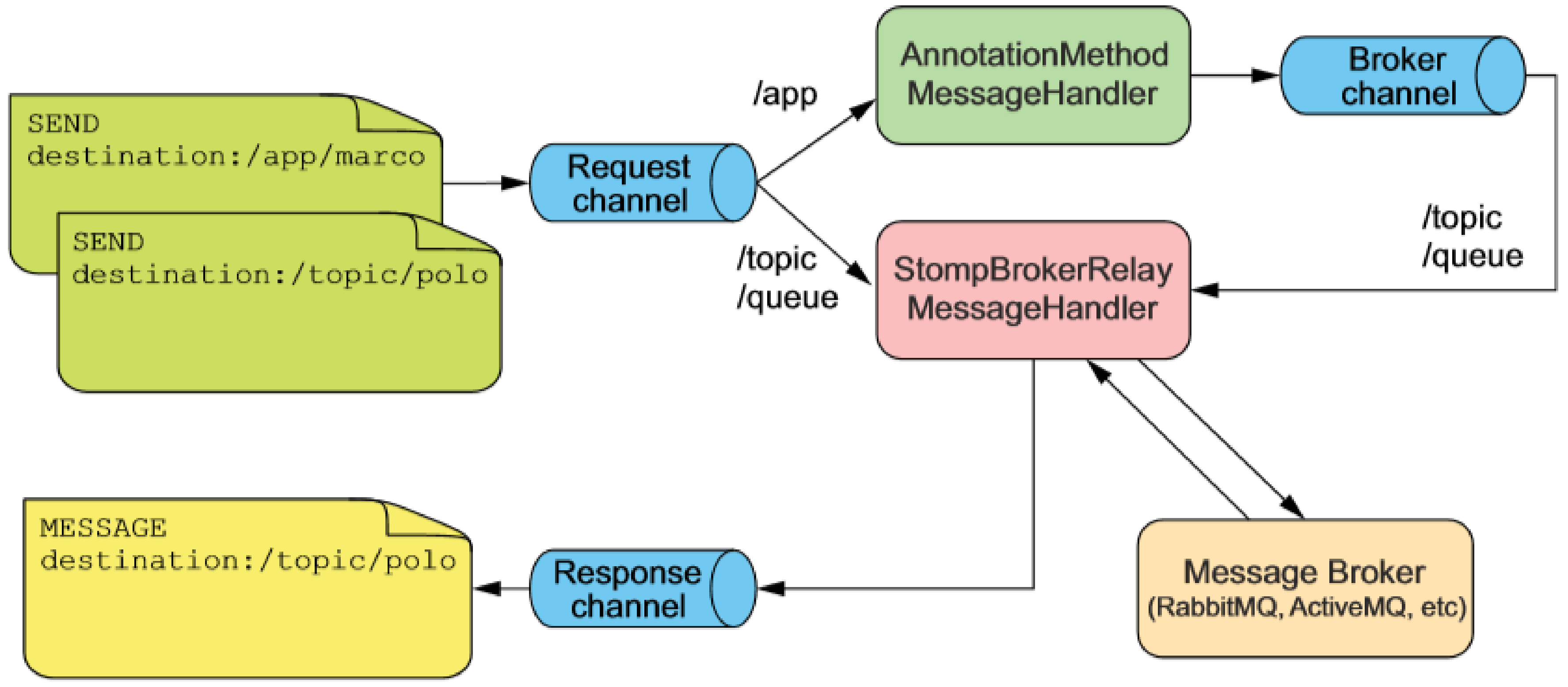


Figure 18.3 **The STOMP broker relay delegates to a real message broker for handling STOMP messages.**

```
@Override
public void configureMessageBroker(MessageBrokerRegistry registry) {
  registry.enableStompBrokerRelay("/topic", "/queue");
  registry.setApplicationDestinationPrefixes("/app");
}
```

Enabling a STOMP broker relay

- Multiple destination and application prefixes possible

```
@Override
public void configureMessageBroker(MessageBrokerRegistry registry) {
  registry.enableStompBrokerRelay("/topic", "/queue");
  registry.setApplicationDestinationPrefixes("/app", "/foo");
}
```

- Changing the default configuration:

```
@Override
public void configureMessageBroker(MessageBrokerRegistry registry) {
    registry.enableStompBrokerRelay("/topic", "/queue")
            .setRelayHost("rabbit.someotherserver")
            .setRelayPort(62623)
            .setClientLogin("marcopolo")
            .setClientPasscode("letmein01");
    registry.setApplicationDestinationPrefixes("/app", "/foo");
  }
```

Handling STOMP messages from the client

Listing 18.6 @MessageMapping handles STOMP messages in a controller.

```
package marcopolo;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.stereotype.Controller;

@Controller
public class MarcoController {

  private static final Logger logger =
      LoggerFactory.getLogger(MarcoController.class);

  @MessageMapping("/marco")
   public void handleShout(Shout incoming) {
    logger.info("Received message: " + incoming.getMessage());
  }

}
```

Handle messages for /app/marco destination

```
public class Shout {
  private String message;

  public String getMessage() {
    return message;
  }

  public void setMessage(String message) {
    this.message = message;
  }
}
```

Message converter

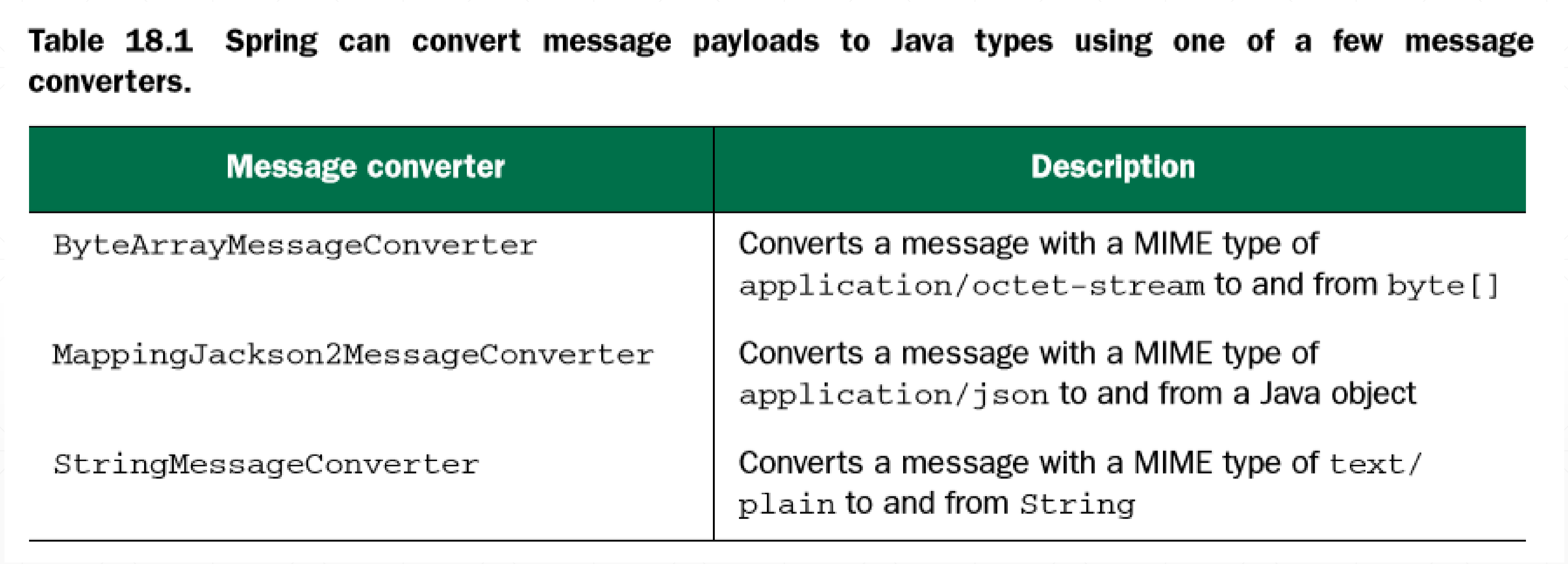
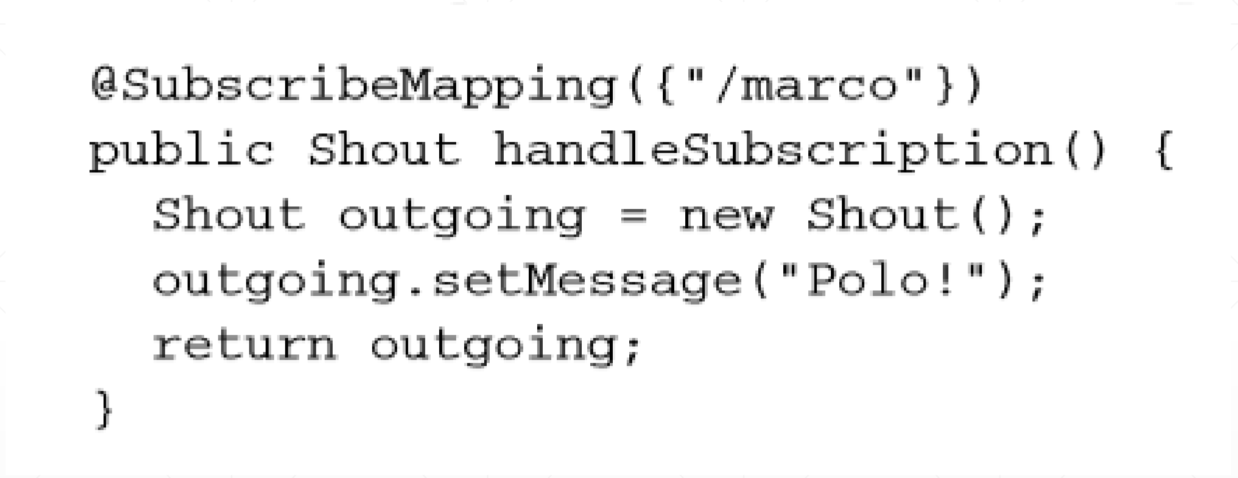


Table 18.1 Spring can convert message payloads to Java types using one of a few message converters.

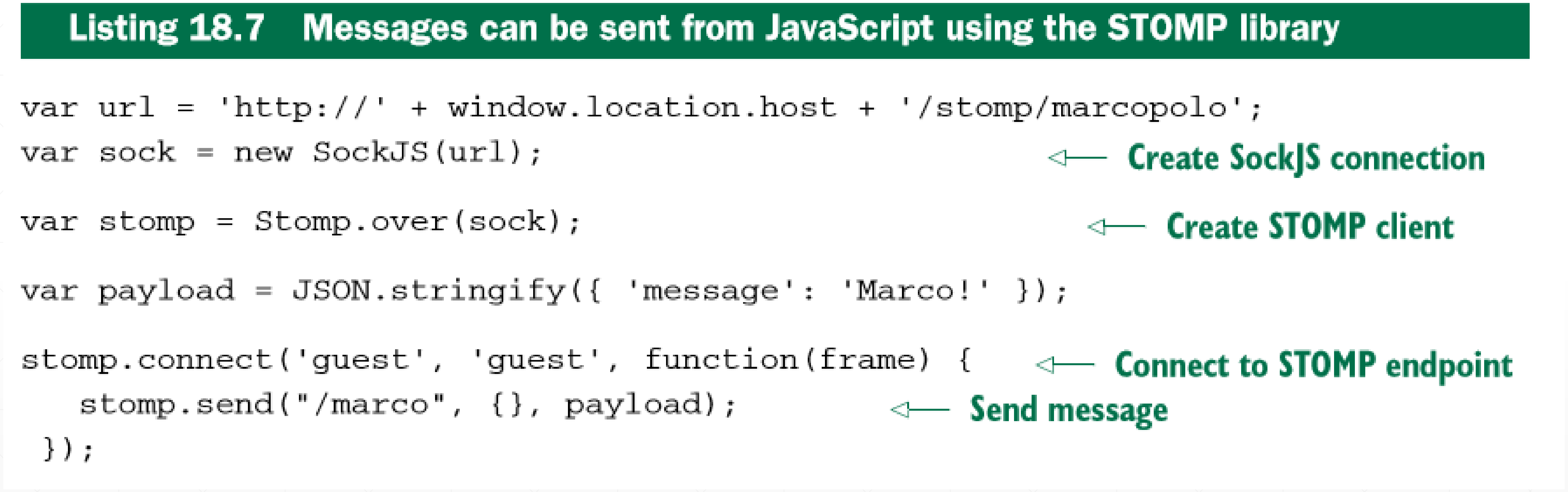
Message converter	Description
`ByteArrayMessageConverter`	Converts a message with a MIME type of `application/octet-stream` to and from `byte[]`
`MappingJackson2MessageConverter`	Converts a message with a MIME type of `application/json` to and from a Java object
`StringMessageConverter`	Converts a message with a MIME type of `text/plain` to and from `String`

Processing subscriptions



```
@SubscribeMapping({"/marco"})
public Shout handleSubscription() {
  Shout outgoing = new Shout();
  outgoing.setMessage("Polo!");
  return outgoing;
}
```

Writing the JavaScript client



Listing 18.7 Messages can be sent from JavaScript using the STOMP library

```
var url = 'http://' + window.location.host + '/stomp/marcopolo';
var sock = new SockJS(url);

var stomp = Stomp.over(sock);

var payload = JSON.stringify({ 'message': 'Marco!' });

stomp.connect('guest', 'guest', function(frame) {
   stomp.send("/marco", {}, payload);
 });
```

Create SockJS connection

Create STOMP client

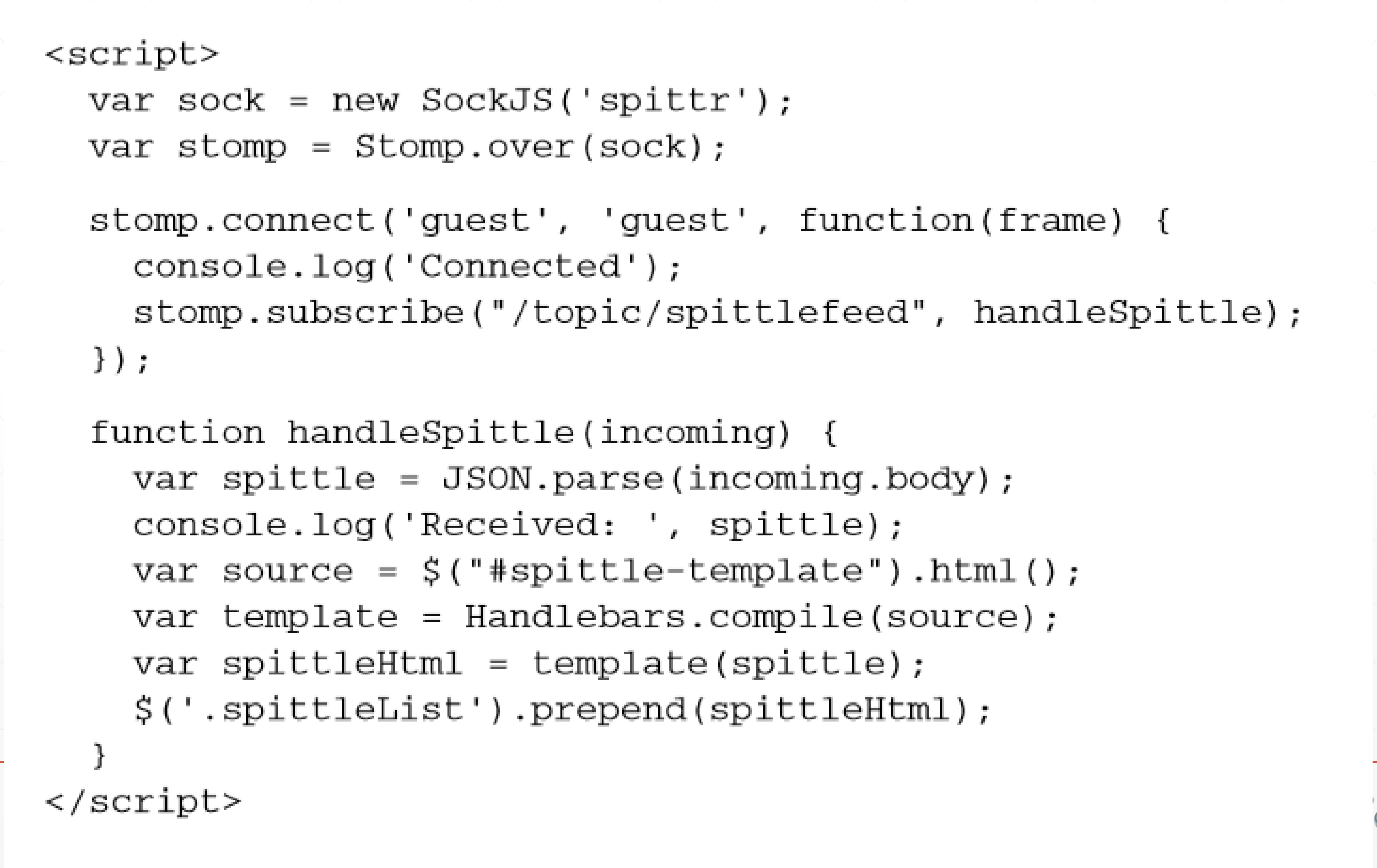
Connect to STOMP endpoint

Send message

Sending a message after handling a message

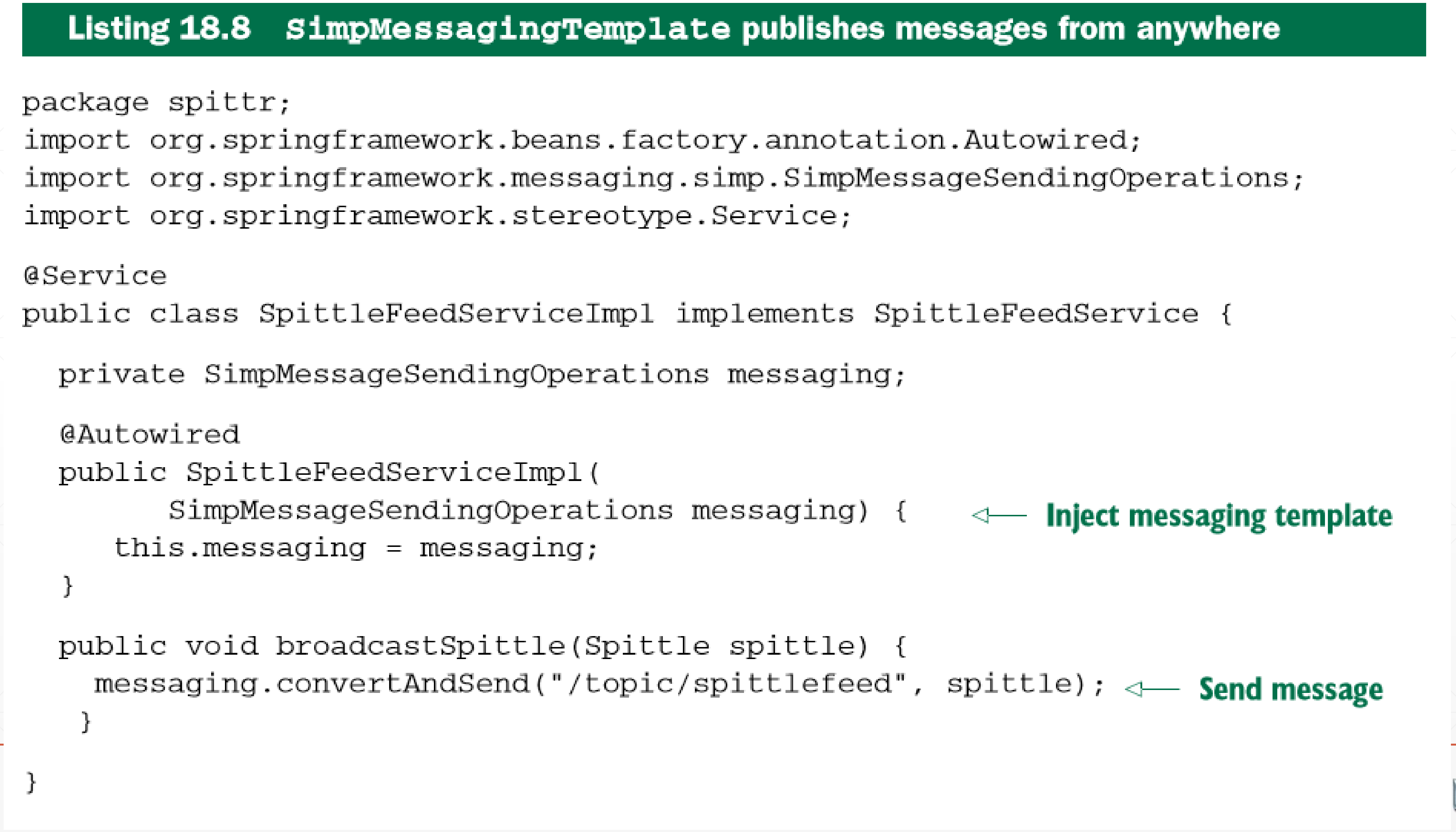
```
@MessageMapping("/marco")
@SendTo("/topic/shout")
public Shout handleShout(Shout incoming) {
  logger.info("Received message: " + incoming.getMessage());

  Shout outgoing = new Shout();
  outgoing.setMessage("Polo!");
  return outgoing;
}
```

Sending a message from anywhere



```
<script>
  var sock = new SockJS('spittr');
  var stomp = Stomp.over(sock);

  stomp.connect('guest', 'guest', function(frame) {
    console.log('Connected');
    stomp.subscribe("/topic/spittlefeed", handleSpittle);
  });

  function handleSpittle(incoming) {
    var spittle = JSON.parse(incoming.body);
    console.log('Received: ', spittle);
    var source = $("#spittle-template").html();
    var template = Handlebars.compile(source);
    var spittleHtml = template(spittle);
    $('.spittleList').prepend(spittleHtml);
  }
</script>
```

Sending a message from anywhere



Listing 18.8 `SimpMessagingTemplate` publishes messages from anywhere

```
package spittr;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.simp.SimpMessageSendingOperations;
import org.springframework.stereotype.Service;

@Service
public class SpittleFeedServiceImpl implements SpittleFeedService {

  private SimpMessageSendingOperations messaging;

  @Autowired
  public SpittleFeedServiceImpl(
        SimpMessageSendingOperations messaging) {      <— Inject messaging template
     this.messaging = messaging;
  }

  public void broadcastSpittle(Spittle spittle) {
    messaging.convertAndSend("/topic/spittlefeed", spittle);  <— Send message
   }

}
```

Working with user-targeted messages

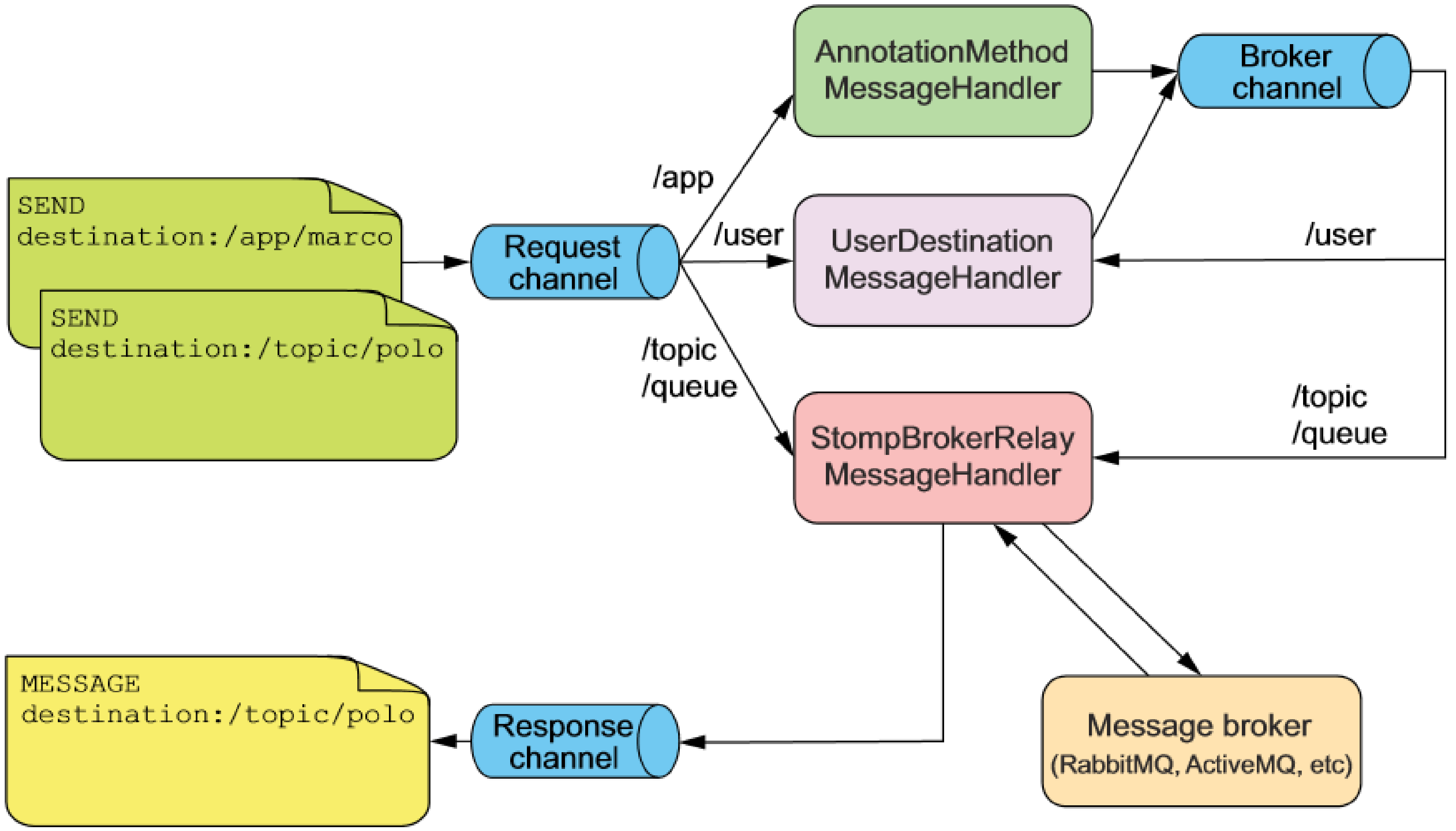


Figure 18.4 **User messages flow through `UserDestinationMessageHandler`, which reroutes them to a destination that's unique to a user.**

Sending messages to a specific user

Listing 18.9 `convertAndSendToUser()` can send a message to a specific user

```
package spittr;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.stereotype.Service;

@Service
public class SpittleFeedServiceImpl implements SpittleFeedService {

  private SimpMessagingTemplate messaging;
  private Pattern pattern = Pattern.compile("\\@(\\S+)");      <- Regex pattern for user mention

  @Autowired
  public SpittleFeedServiceImpl(SimpMessagingTemplate messaging) {
    this.messaging = messaging;
  }

  public void broadcastSpittle(Spittle spittle) {

    messaging.convertAndSend("/topic/spittlefeed", spittle);

    Matcher matcher = pattern.matcher(spittle.getMessage());
    if (matcher.find()) {
      String username = matcher.group(1);
      messaging.convertAndSendToUser(                          <- Send notification to user
           username, "/queue/notifications",
          new Notification("You just got mentioned!"));
    }
```

Creating REST APIs with Spring MVC