

---

# Using WebSocket with Spring

# 目錄

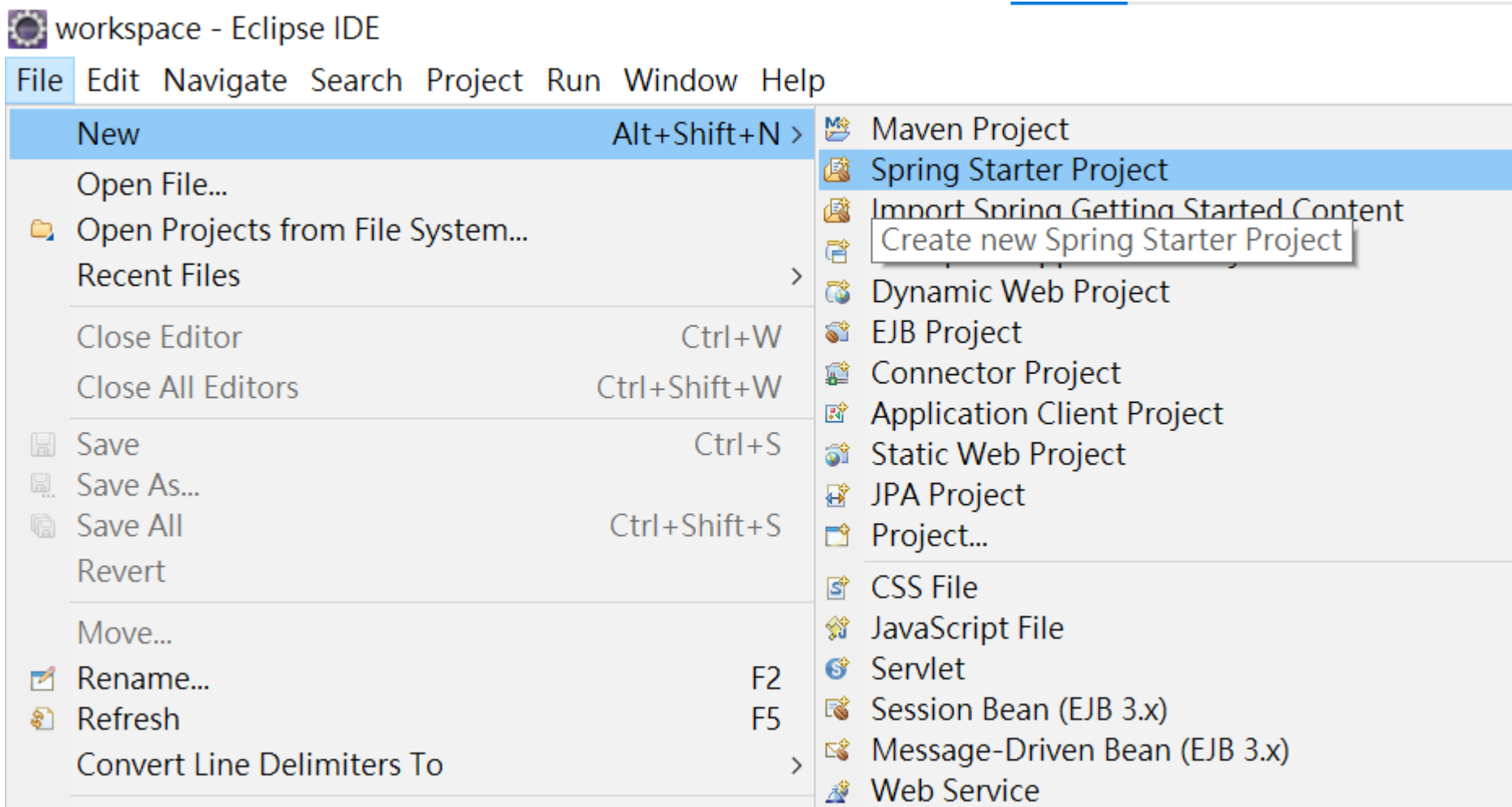
---



一、建立具備WebSocket功能的Spring Boot專案	p-003
二、發送與接收訊息的低階API	p-017
三、SockJS	p-037
四、使用STOMP訊息	p-045
五、聊天室	p-065
六、附錄	p-073

---

# 一、建立具備WebSocket功能的 Spring Boot專案

# 新建Spring Boot專案





## New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

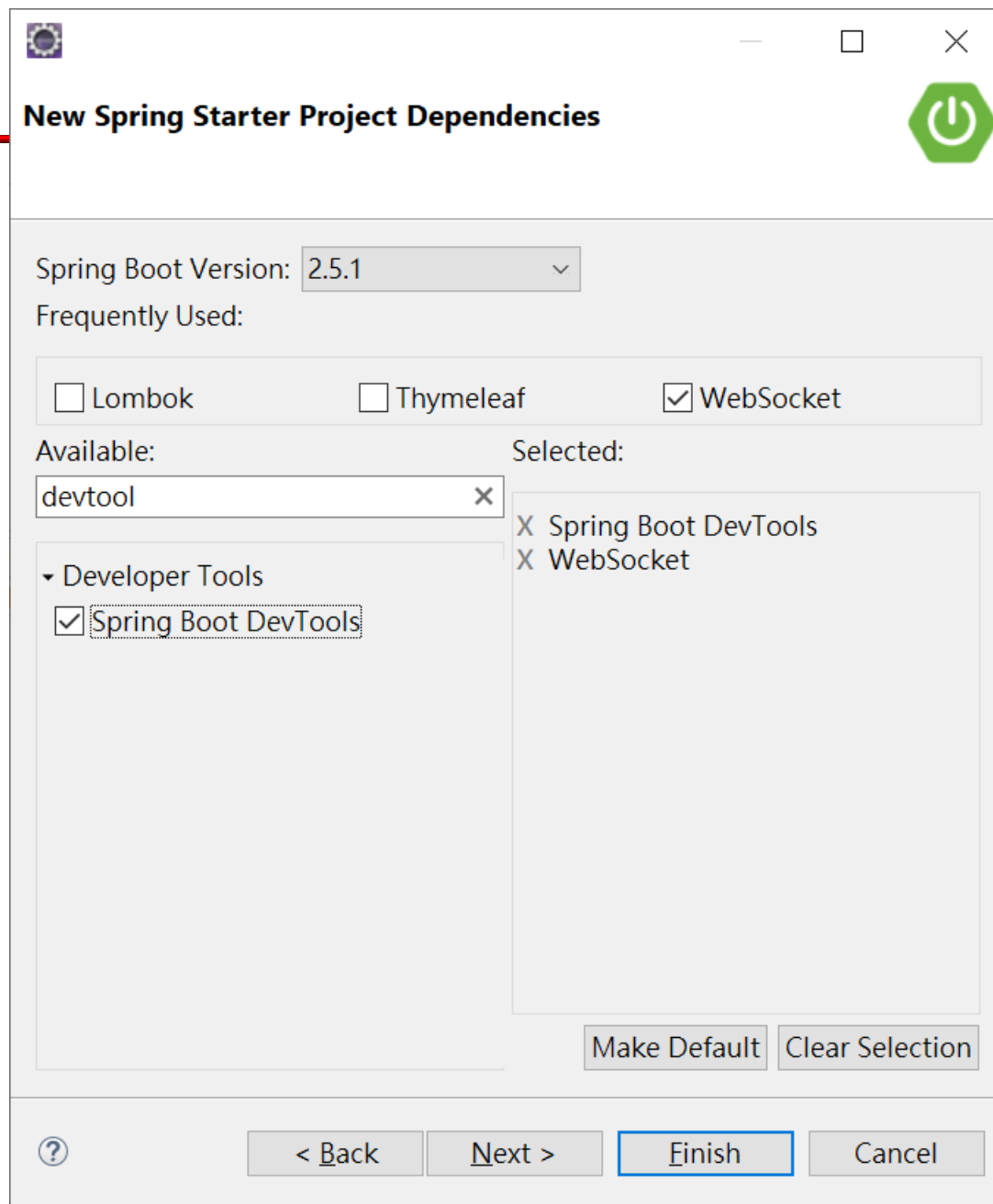
Package:

Working sets

☐ Add project to working sets

Working sets:

### 一、具備WebSocket功能的專案



The image shows a 'New Spring Starter Project Dependencies' dialog box. At the top, there's a title bar with a gear icon, standard window controls, and a green power button icon. The main content area is divided into sections. The first section has 'Spring Boot Version: 2.5.1' with a dropdown arrow. Below it is 'Frequently Used:' with three checkboxes: 'Lombok' (unchecked), 'Thymeleaf' (unchecked), and 'WebSocket' (checked). The next section is split into 'Available:' and 'Selected:'. Under 'Available:', there's a search box containing 'devtool' and a list under 'Developer Tools' with 'Spring Boot DevTools' checked. Under 'Selected:', there's a list with 'Spring Boot DevTools' and 'WebSocket', both preceded by an 'X' icon. At the bottom right of the main area are 'Make Default' and 'Clear Selection' buttons. The footer contains a help icon, '< Back' button, 'Next >' button, 'Finish' button (highlighted with a blue border), and 'Cancel' button.

**New Spring Starter Project Dependencies**

Spring Boot Version: 2.5.1

Frequently Used:

☐ Lombok ☐ Thymeleaf ☒ WebSocket

Available: Selected:

devtool

▼ Developer Tools

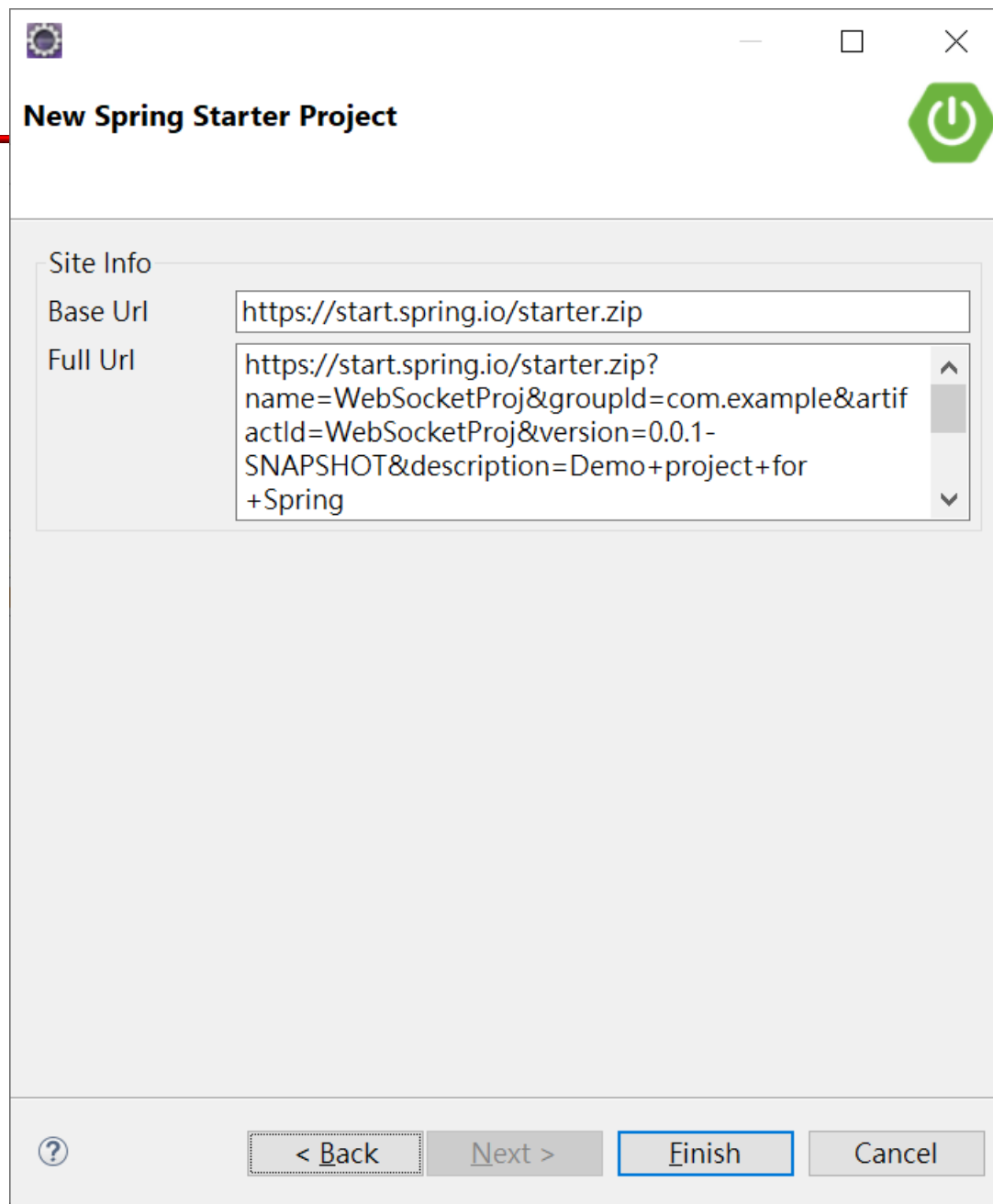
☒ Spring Boot DevTools

X Spring Boot DevTools  
X WebSocket

Make Default Clear Selection

? < Back Next > Finish Cancel

一、具備WebSocket功能的專案



The image shows a 'New Spring Starter Project' dialog box. It has a title bar with a gear icon, a maximize button, and a close button. A green power button icon is in the top right corner. The main area is titled 'Site Info' and contains two text fields: 'Base Url' with the value 'https://start.spring.io/starter.zip' and 'Full Url' with a longer URL including project details. At the bottom, there are four buttons: a help button (question mark), '< Back', 'Next >', and 'Finish' (which is highlighted with a blue border). A 'Cancel' button is also present.

**New Spring Starter Project**

Site Info

Base Url:

Full Url:

? < Back Next > Finish Cancel

一、具備WebSocket功能的專案

# pom.xml 原始內容

---

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.1</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.example</groupId>
  <artifactId>WebSocketProj</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>WebSocketProj</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>11</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-websocket</artifactId>
    </dependency>
```



# pom.xml 原始內容

---

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

</project>
```

# 使用JSP加入下列標籤

---

```
<dependency>  
  <groupId>org.apache.tomcat.embed</groupId>  
  <artifactId>tomcat-embed-jasper</artifactId>  
</dependency>
```

```
<dependency>  
  <groupId>javax.servlet</groupId>  
  <artifactId>jstl</artifactId>  
</dependency>
```

# 前端技術需加入下列標籤

---

```
<dependency>  
  <groupId>org.webjars</groupId>  
  <artifactId>webjars-locator-core</artifactId>  
</dependency>
```

```
<dependency>  
  <groupId>org.webjars</groupId>  
  <artifactId>sockjs-client</artifactId>  
  <version>1.0.2</version>  
</dependency>
```

```
<dependency>  
  <groupId>org.webjars</groupId>  
  <artifactId>stomp-websocket</artifactId>  
  <version>2.3.3</version>  
</dependency>
```

# 前端技術需加入下列標籤

..

```
<dependency>  
  <groupId>org.webjars</groupId>  
  <artifactId>bootstrap</artifactId>  
  <version>3.3.7</version>  
</dependency>
```

```
<dependency>  
  <groupId>org.webjars</groupId>  
  <artifactId>jquery</artifactId>  
  <version>3.1.1-1</version>  
</dependency>
```

# 修改application.properties

---

- 加入下列敘述

```
server.port=8080
```

```
server.servlet.context-path=/websocket
```

```
spring.mvc.view.prefix=/WEB-INF/views/
```

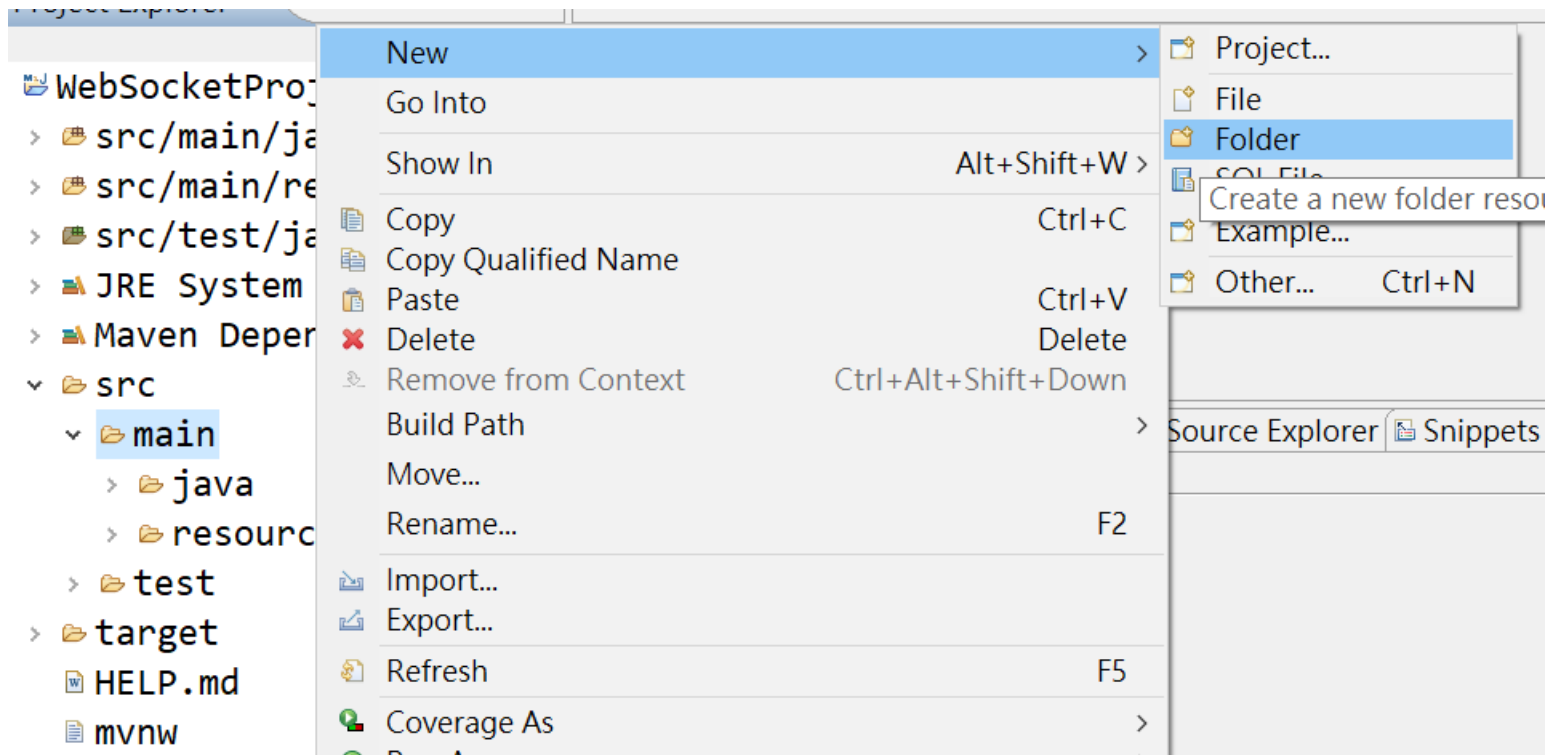
```
spring.mvc.view.suffix=.jsp
```

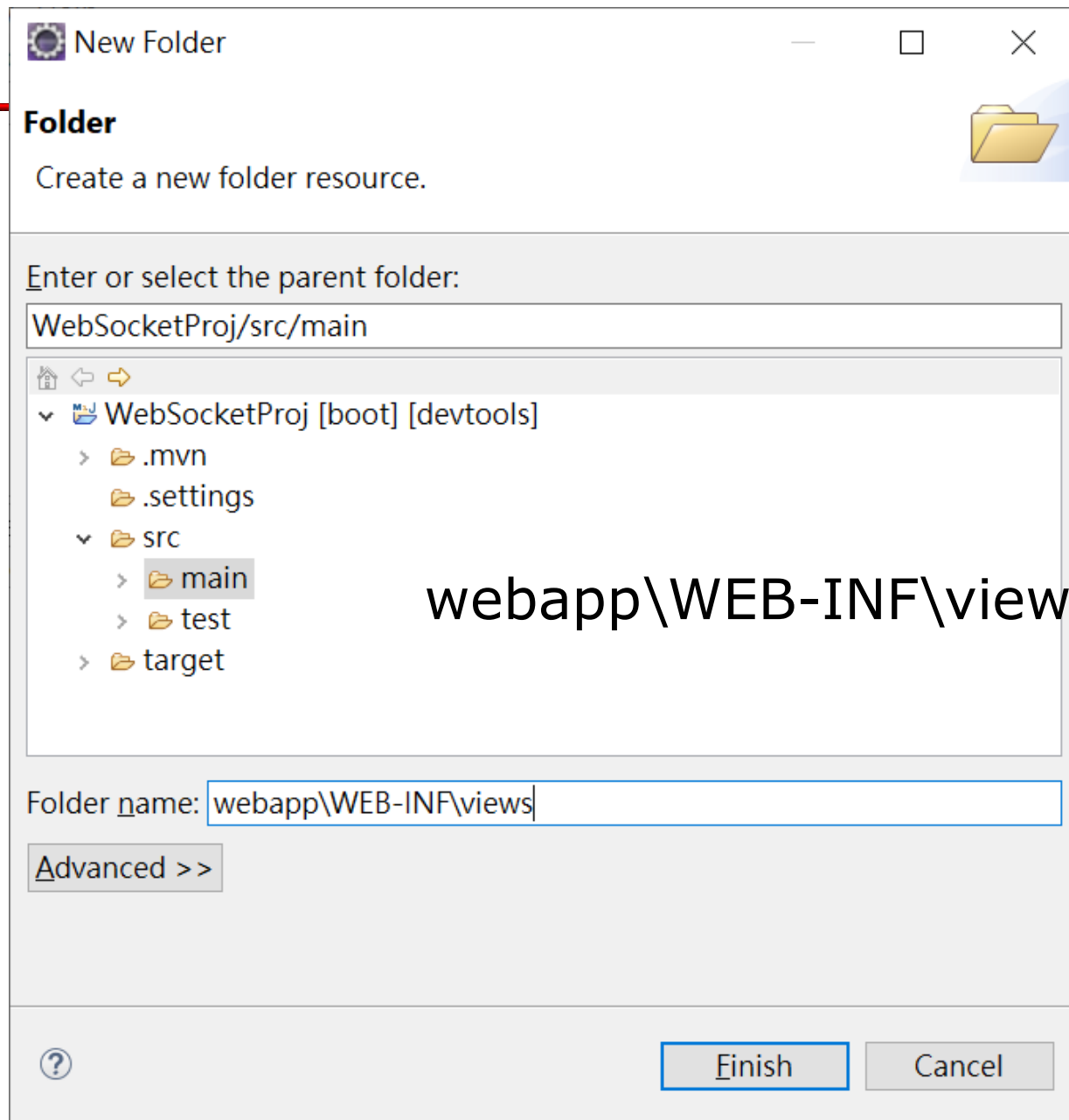
```
# Show Messages related to WebSocket
```

```
logging.level.org.springframework.messaging=trace
```

```
logging.level.org.springframework.web.socket=trace
```

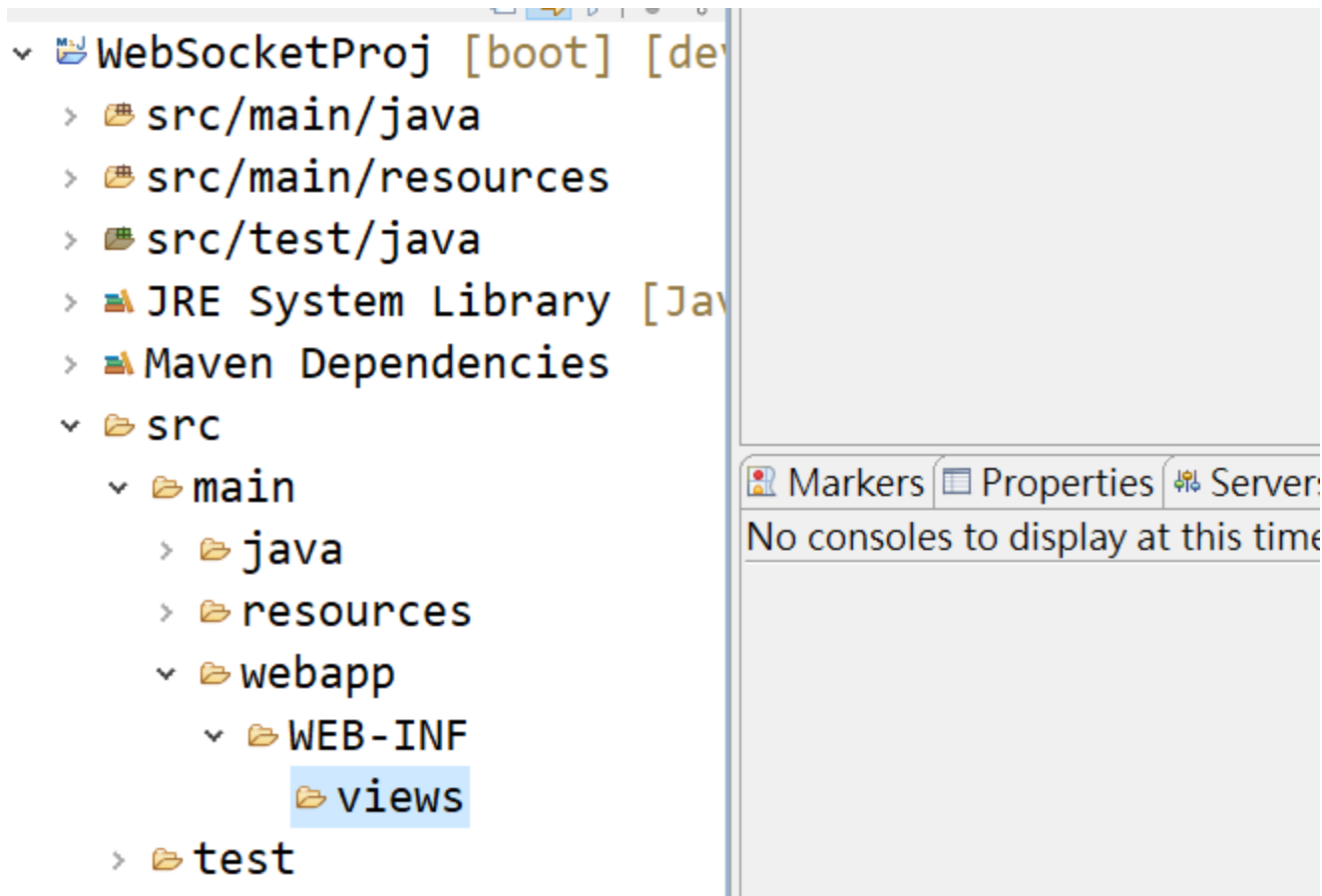
# 建立/WEB-INF/views/資料夾





webapp\WEB-INF\views

# 完成





---

## 二、發送與接收訊息的低階API

# 相關的類別與介面

---

- Spring框架提供的o.s.w.s.WebSocketHandler介面可讓伺服器 and 瀏覽器間以WebSocket進行連線與傳遞即時資訊。
- Spring框架提供實作此介面的抽象類別AbstractWebSocketHandler，我們只需要繼承此類別，Override必要的方法即可。

# WebSocketHandler 介面

---

- *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();*
- 第一、二與四個方法為WebSocket連線之生命週期相關方法
- 實務上處理前端請求的Handler類別會繼承已經實作上述介面之抽象類別：o.s.w.s.h.AbstractWebSocketHandler

# AbstractWebSocketHandler抽象類別

// 有關handleMessage()的內容

@Override

```
public void handleMessage(WebSocketSession session, WebSocketMessage<?> message) throws Exception {  
    if (message instanceof TextMessage) {  
        handleTextMessage(session, (TextMessage) message);  
    }  
    else if (message instanceof BinaryMessage) {  
        handleBinaryMessage(session, (BinaryMessage) message);  
    }  
    else if (message instanceof PongMessage) {  
        handlePongMessage(session, (PongMessage) message);  
    }  
    else {  
        throw new IllegalStateException("Unexpected WebSocket message type: " + message);  
    }  
}
```

- 繼承此類別時只需要依照處理資料的類型Override下列三方法中的一個即可：

protected void handleTextMessage(WebSocketSession session, TextMessage message) throws Exception

protected void handleBinaryMessage(WebSocketSession session, BinaryMessage message) throws Exception

protected void handlePongMessage(WebSocketSession session, PongMessage message) throws Exception

上面三個方法只會處理一種類型的資料

# MacroHandler.java

---

```
package com.example.demo.websocket._01.handler;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Set;

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

public class MacroHandler extends AbstractWebSocketHandler {
    Logger logger = LoggerFactory.getLogger(getClass());

    @Override
    protected void handleTextMessage(WebSocketSession session,
        TextMessage message) throws Exception {
        logger.info("收到訊息" + message.getPayload() + ", From=" + session.getId());
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss SSS");
        session.sendMessage(new TextMessage("現在時間:" + sdf.format(new Date())));
    }
}
```

# MacroHandler.java ..

---

@Override

```
public void afterConnectionEstablished(WebSocketSession session) throws Exception {  
    logger.info("建立新連線: id=" + session.getId()  
        + ", getRemoteAddress()" + session.getRemoteAddress());
```

```
    HttpHeaders headers = session.getHandshakeHeaders();
```

```
    Set<String> set = headers.keySet();
```

```
    for(String key : set) {
```

```
        Object o = headers.get(key);
```

```
        logger.info("請求標頭: " + key + "==>" + o);
```

```
    }
```

```
}
```

@Override

```
public void afterConnectionClosed(WebSocketSession session, CloseStatus status)  
    throws Exception {
```

```
    logger.info("關閉連線: id=" + session.getId()
```

```
        + ", getRemoteAddress()" + session.getRemoteAddress());
```

```
}
```

```
}
```

# 處理文字訊息

---

protected void handleMessage  
(WebSocketSession session,  
TextMessage message) throws Exception

- 接收客戶端送來的訊息
  - TextMessage類別的getPayload()方法
    - TextMessage類別：代表WebSocket上傳送的文字訊息
- 回覆給客戶端的訊息
  - WebSocketSession介面的sendMessage(TextMessage message)方法

# WebSocketSession 介面

---

- 經由此介面可以傳送訊息並取得與連線有關的資訊。
- `String getId();`
  - 取得 Session Id
- `void sendMessage(WebSocketMessage<?> message) throws IOException;`
  - 傳送訊息
- `void close() throws IOException;`
  - 關閉連線
- `HttpHeaders getHandshakeHeaders();`
  - 取得第一次連線時的 Http 請求標頭



# WebSocket連線 vs Http連線

---

- WebSocket

- 每次客戶端送出WebSocket連線要求、並且連線成功後會一直保持連線，直到任一方決定斷線為止。
- 於連線期間，雙方可以同時向對方發送訊息及接收對方送來的訊息。
- 每次關閉連線(無論正常關閉或因發生例外而關閉)後而重新建立連線時都會重新建立一個WebSocketSession物件。一次連線就是一個WebSocketSession。

- Http

- 每次客戶端送出請求前會建立連線，客戶端收到回應後會中斷連線。
- 於連線期間，客戶端先送出要求資源的請求，伺服器端才會發出回應。一次連線可進行一次的『請求/回應』(HTTP1.0)或一次連線可進行多次的『請求/回應』(HTTP1.1的持續性連線)
- 同一個使用者於一段時間的多次請求形成一個HttpSession。

# 連線建立之後

---

public void afterConnectionEstablished  
(WebSocketSession session) throws Exception

```
logger.info("建立新連線: id=" + session.getId()  
    + ", getRemoteAddress()=" + session.getRemoteAddress());
```

```
HttpHeaders headers = session.getHandshakeHeaders();
```

```
Set<String> set = headers.keySet();
```

```
for(String key : set) {
```

```
    Object o = headers.get(key);
```

```
    logger.info(" 請求標頭: " + key + "==>" + o);
```

```
}
```

- session.getId(): WebSocket準備的Session Id
- session.getHandshakeHeaders(): 取得Handshake時的Http請求標頭

# 執行結果

```
建立新連線: id=59f9ca09-0518-ecc3-8223-edccdfb3cc9d, getRemoteAddress()=/0:0:0:0:0:0:0:1:63426
請求標頭: host==>[localhost:8080]
請求標頭: connection==>[Upgrade]
請求標頭: pragma==>[no-cache]
請求標頭: cache-control==>[no-cache]
請求標頭: user-agent==>[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/91.0.4472.114 Safari/537.36]
請求標頭: upgrade==>[websocket]
請求標頭: origin==>[http://localhost:8080]
請求標頭: sec-websocket-version==>[13]
請求標頭: accept-encoding==>[gzip, deflate, br]
請求標頭: accept-language==>[zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7,zh-CN;q=0.6]
請求標頭: cookie==>[JSESSIONID=A7ABBBF021924ED332BD4BF7BD301AEF]
請求標頭: sec-websocket-key==>[zzl1eiNL7Vy/cu9MfTgGZw==]
請求標頭: sec-websocket-extensions==>[permessage-deflate; client_max_window_bits]
// 以下為Handler方法丟出之訊息
Handling TextMessage payload=[Hello, 你好], byteCount=14, last=true] in
StandardWebSocketSession[id=59f9ca09-0518-ecc3-8223-edccdfb3cc9d,
uri=ws://localhost:8080/websocket/macro]
收到訊息=Hello, 你好, From=59f9ca09-0518-ecc3-8223-edccdfb3cc9d
Sending TextMessage payload=[現在時間: 2021..], byteCount=37, last=true],
StandardWebSocketSession[id=59f9ca09-0518-ecc3-8223-edccdfb3cc9d,
uri=ws://localhost:8080/websocket/macro]
```

# Handshake時的Http請求標頭

## ▼ Request Headers [View source](#)

Accept-Encoding: gzip, deflate, br

Accept-Language: zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7,zh-CN;q=0.6

Cache-Control: no-cache

Connection: Upgrade

Cookie: JSESSIONID=A7AB8BF021924ED332BD4BF78D301AEF

Host: localhost:8080

Origin: http://localhost:8080

Pragma: no-cache

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

Sec-WebSocket-Key: LnawGkGIOeD6LF6JsiyznA==

Sec-WebSocket-Version: 13

Upgrade: websocket

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome

# 關閉連線

---

*@Override*

```
public void afterConnectionClosed (WebSocketSession session,  
    CloseStatus status) throws Exception {
```

```
    logger.info("關閉連線: id=" + session.getId()  
        + ", getRemoteAddress()" + session.getRemoteAddress());  
}
```

- 執行結果：

關閉連線: id=59f9ca09-0518-ecc3-8223-edccdfb3cc9d, getRemoteAddress()=/0:0:0:0:0:0:0:1:56778

# Server端的配置工作

```
package com.example.demo.websocket._01.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.socket.config.annotation.EnableWebSocket;
import org.springframework.web.socket.config.annotation.WebSocketConfigurer;
import org.springframework.web.socket.config.annotation.WebSocketHandlerRegistry;
import com.example.demo.websocket._01.handler.MacroHandler;
```

**@Configuration("WebSocketConfig1")**

**@EnableWebSocket**

```
public class WebSocketConfig implements WebSocketConfigurer {
```

```
    @Override
```

```
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
```

```
        // 請求路徑 /macro 會映射到 MacroHandler 類別
```

```
        registry.addHandler(macroHandler(), "/macro");
```

```
    }
```

"ws://" + window.location.host + contextPath + **"/macro"**

```
    @Bean
```

```
    public MacroHandler macroHandler() {
```

```
        return new MacroHandler();
```

```
    }
```

```
}
```

# 前端程式 \_01\_basicws.jsp .

---

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
<link rel='stylesheet' href="<c:url value='/css/style.css' />" />
<meta charset="UTF-8">
<title>Spring Boot</title>
<script>
let contextPath = '${contextPath}';
let sock = null;
let timer = null;
let messageArea = null;
window.onload = function(){
    let btnStart = document.getElementById("btnStart");
    let btnEnd = document.getElementById("btnEnd");
    let url = "ws://" + window.location.host + contextPath + '/macro' ;
    btnEnd.disabled = true;
    messageArea = document.getElementById("messageArea");
```

# 前端程式 \_01\_basicws.jsp ..

---

```
btnStart.addEventListener('click', function(){
    sock = new WebSocket(url);
    btnEnd.disabled = false;
    btnStart.disabled = true;
    sock.onopen = function (){
        console.log('開啟WebSocket連線');
        messageArea.value += '開啟WebSocket連線\n';
        sayMarco();
    }
    // 讀取後端程式傳來之訊息
    sock.onmessage = function(e){
        console.log('收到訊息', e.data);
        messageArea.value += '收到訊息:' + e.data + "\n";
        timer = setTimeout(function(){sayMarco(); }, 2000)
    }

    sock.onclose = function (){
        clearTimeout(timer);
        console.log('關閉WebSocket連線');
        messageArea.value += '關閉WebSocket連線\n';
    }
});
```

二、發送與接收訊息的低階API



# 前端程式 \_01\_basicws.jsp ...

---

```
btnEnd.addEventListener('click', function(){
    sock.close();
    btnEnd.disabled = true;
    btnStart.disabled = false;
});
}

function sayMarco(){
    console.log('傳送訊息...');
    sock.send('Hello，你好'); // 將前端資料傳去給後端
}

function clearText(){
    messageArea.value = "";
}

</script>
</head>
<body>
<div align='center'>
    <h2>初等Spring WebSocket用法</h2>
```

# 前端程式 \_01\_basicws.jsp ....

[illegible]

# 前端程式摘要

---

- 建立與後端程式的連線

```
let url = "ws://" + window.location.host + contextPath + '/macro' ;  
sock = new WebSocket(url);      // /macro為 Endpoint
```

- 將前端資料傳給後端

```
sock.send('傳送的文字資料');
```

- 編寫與生命週期有關的方法

```
sock.onopen, sock.onmessage, sock.onclose, sock.onerror
```

- 讀取後端傳來的資料

```
sock.onmessage = function(e){  
    console.log('收到訊息', e.data);  
}
```

# 執行結果

## 初等Spring WebSocket用法

開始

結束

開啟WebSocket連線

收到訊息:現在時間: 2021-06-16 05:13:17 019

收到訊息:現在時間: 2021-06-16 05:13:19 028

收到訊息:現在時間: 2021-06-16 05:13:21 034

收到訊息:現在時間: 2021-06-16 05:13:23 050

關閉WebSocket連線

清除訊息

[回前頁](#)

---

## 三、SockJS

當瀏覽器不支援WebSocket時

# 瀏覽器對WebSocket的支援

---

- WebSocket是一個較新的技術，然而並非所有瀏覽器都支援此技術，因此前端程式需要一種備用方案以便前後端在任何情況下都能使用WebSocket技術。
- 在WebSocket尚未普及化的最佳選擇：SockJS。

# SockJS

---

- SockJS協定讓所有瀏覽器可以使用相同的 API 做到即時通訊，相關方法的名稱基本上和 WebSocket相同。
- 它採用混合方式解決瀏覽器功能不一致的問題，對外的介面完全模擬WebSocket，而內部實作則採用非常智慧的做法：如果能使用瀏覽器提供之WebSocket則採用之，否則就會選擇其他的實作模擬出即時通訊。
- 使用SockJS Client的前端程式只需要專注在幾個重要的函數上而不需考慮瀏覽器的相容性。
- 前端與後端程式都需要修改才能使用SockJS。

# 在Server端啟用SockJS功能

---

- 呼叫WebSocketHandlerRegistration介面提供的withSockJS()方法：

```
registry.addHandler(macroHandler(), "/macro").withSockJS();
```

```
registry.addHandler(macroHandlerSockJS(), "/macroSockJS").withSockJS();
```



# 在Client端啟用SockJS功能

---

- 要在客戶端使用SockJS，必須確定前端程式載入了SockJS客戶端程式庫：

```
<script src="<c:url value='/webjars/sockjs-client/sockjs.min.js' />"></script>  
<script src="<c:url value='/webjars/sockjs-client/1.0.2/sockjs.min.js' />"></script>
```

- 專案的pom.xml應包含下列標籤：

```
<dependency>  
  <groupId>org.webjars</groupId>  
  <artifactId>sockjs-client</artifactId>  
  <version>1.0.2</version>  
</dependency>
```

# Javascript程式碼

---

```
<script type="text/javascript">  
    var contextPath = "${contextPath}";  
</script>  
// 以下為 app.js的內容  
var url = contextPath + '/springio_websocket_example';  
var socket = new SockJS(url);
```

url若包含通訊協定，它必須是http://或https://而不能是ws://或ws://

'/springio\_websocket\_example': 視實際情況編寫

# MacroHandlerSockJS類別

---

- 與前一章之MacroHandler之完全相同，不需須改。

# WebSocketConfig類別

---

```
package com.example.demo.websocket._02.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.socket.config.annotation.EnableWebSocket;
import org.springframework.web.socket.config.annotation.WebSocketConfigurer;
import org.springframework.web.socket.config.annotation.WebSocketHandlerRegistry;
import com.example.demo.websocket._02.handler.MacroHandlerSockJS;
```

```
@Configuration("WebSocketConfigSockJS")
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {
    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
        // 請求路徑 /macroSockJS 會映射到 MacroHandler 類別
        registry.addHandler(macroHandlerSockJS(), "/macroSockJS").withSockJS();
    }
    @Bean
    public MacroHandlerSockJS macroHandlerSockJS() {
        return new MacroHandlerSockJS();
    }
}
```

---

## 四、使用STOMP訊息

# WebSocket的缺點

---

- 直接使用WebSocket進行連線類似在TCP協定下直接進行Socket連線。資料是以一連串的位元組來傳送而不會提供有關如何傳送或如何處理它的任何訊息。由於沒有更高階的協定，除非自行編寫額外的程式碼，否則直接使用Socket連線進行較複雜的應用將變得很困難。
- 類似HTTP協定在TCP協定上增加『請求』與『回應』的模型層，STOMP在WebSocket之上提供一基於框架的線格式(frame-based wire format)用以定義訊息的格式。有了這樣的協定，連線的雙方要表達所做的事情就有一致性的標準。
  - Frame-Based: If I told you that I'm sending you 8 bytes and I send you 6 bytes, you would wait for the next 2 bytes and then say "this is a message".
  - Stream-Based: I send you 6 bytes. I send you 2 bytes. Is it one message? two messages? six messages? How do you know where one message starts and another begins?

# STOMP

---

- STOMP: Simple (or Streaming) Text Oriented Message Protocol
- STOMP 提供了一種可互操作的有線格式，以便 STOMP 客戶端可以與任何 STOMP 訊息代理進行通信，以在多種語言、平台和代理之間提供簡單而廣泛的消息互操作性。
  - STOMP provides an interoperable wire format so that STOMP clients can communicate with any STOMP message broker to provide easy and widespread messaging interoperability among many languages, platforms and brokers
    - Source: <https://stomp.github.io>

# Interoperable / Interoperability

---

- **Interoperable:** 可互操作的：(電腦系統或軟體)之間能夠交換和利用彼此的訊息。
  - (of computer systems or software) able to exchange and make use of information.
- **Interoperability:** 如果兩個或多個系統使用共同的資料格式和通信協定並能夠相互通信，則它們展現出『互操作性』。XML 和 SQL 是常見數據格式和協議的範例。
  - If two or more systems use a common data formats and communication protocols and are capable of communicating with each other, they exhibit syntactic interoperability. XML and SQL are examples of common data formats and protocols.



# Wire Format

---

- **Wire Format:** 『線格式』只是一個通用術語，表示用於表示記憶體或硬碟上之資料的格式。電線這個詞實際上是指這個儲存的資料最終將會通過電信線路傳送出去("通過電線發送")。
  - The term wire format is just a general term meaning a format for representing data in memory or on a hard disk. The word wire actually refers to the fact that this stored data is eventually sent out on a communications line ("sent out over the wire").

# STOMP Frames

- STOMP 是一種基於Frame的協議，它假設在底層存在一個可靠的雙向流網路協定(例如 TCP)。客戶端和伺服器將通過發送的 STOMP Frame進行通信。Frame的結構如下所示：

COMMAND	命令
header1:value1	標頭1:值1
header2:value2	標頭2:值2
	空白行
Body^@	Payload^@

# STOMP Commands

---

- CONNECT
- SEND
- SUBSCRIBE
- UNSUBSCRIBE
- BEGIN
- COMMIT
- ABORT
- ACK
- NACK
- DISCONNECT
- ERROR
  - 只有SEND, MESSAGE, and ERROR 可以有Body

# 後端程式啟用STOMP功能

---

- 系統必須使用下列註釋來啟用STOMP

## @EnableWebSocketMessageBroker

- 此註釋為Class-level之註釋，必須同時與@Configuration一起作用在類別上。
- 使用此註釋後，以@Controller修飾的控制器類別中使用@MessageMapping註釋標示一個要處理STOMP協定訊息的方法。
- MessageBroker: 訊息代理。

# WebSocketMessageBrokerConfigurer 介面

---

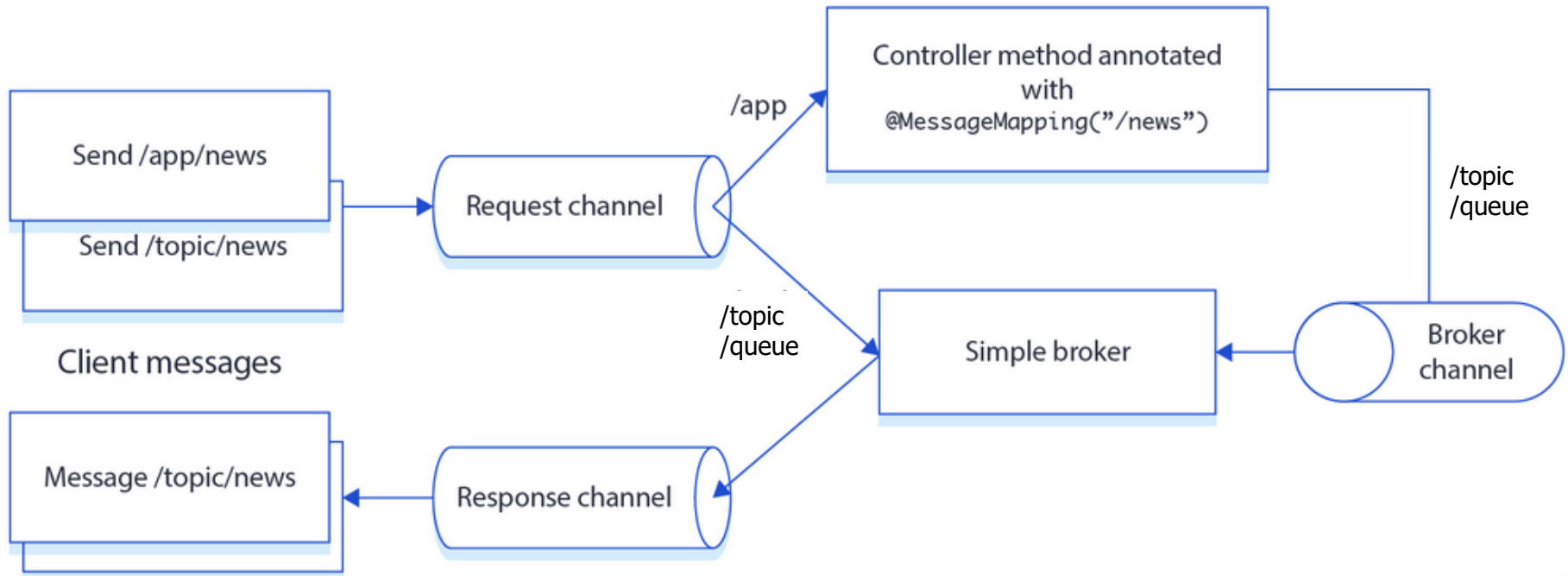
- 介面的registerStompEndpoints()方法定義提供WebSocket服務的端點，並聲明是否支援SockJS()
- 介面的configureMessageBroker()方法定義何種訊息要由哪個方法處理，以及客戶端要訂閱的服務

# 範例

---

```
package com.example.demo.websocket._03.config;
省略import敘述
@Configuration("WebSocketConfig3")
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        // 凡是@SendTo中以"/topic", "/queue"都交由訊息代理處理
        config.enableSimpleBroker("/topic", "/queue");
        // 交給Controller中, @MessageMapping之("/app/...")之控制器方法處理
        config.setApplicationDestinationPrefixes("/app");
    }
    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/gs-guide-websocket")
            .withSockJS();    // 支援SockJS()
    }
}
```

# 範例



# registerStompEndpoints() 方法

- 註冊一個STOMP端點，任何前端程式要由Javascript程式進行STOMP連線時都必須在連線字串中標示此端點，接下來客戶端才能進行訂閱(SUBSCRIBE)或發送(SEND)訊息到目的地。
- 端點在Request URL中的位置：

ws://localhost:8080/mvcExercise/gs-guide-websocket/



The diagram shows the URL `ws://localhost:8080/mvcExercise/gs-guide-websocket/` broken down into five segments by red vertical lines. Below each segment is a label in Chinese and English: 通訊協定 (Protocol), 主機名稱 (Host), 埠號 (Port), 系統名稱 (System Name), and 端點 (Endpoint). The 'Context Path' label is placed below the 'mvcExercise' segment.

通訊協定	主機名稱	埠號	系統名稱	端點
ws://	localhost	:8080/	mvcExercise/	gs-guide-websocket/

Context Path

End Point

http://localhost:8080/mvcExercise/gs-guide-websocket/



The diagram shows the URL `http://localhost:8080/mvcExercise/gs-guide-websocket/` broken down into five segments by red vertical lines. Below each segment is a label in Chinese and English: 通訊協定 (Protocol), 主機名稱 (Host), 埠號 (Port), 系統名稱 (System Name), and 端點 (Endpoint). The 'Context Path' label is placed below the 'mvcExercise' segment.

通訊協定	主機名稱	埠號	系統名稱	端點
http://	localhost	:8080/	mvcExercise/	gs-guide-websocket/

Context Path

End Point



# 處理來自客戶端的訊息

---

@Controller

```
public class GreetingController {
```

```
    // if a message is sent to the /hello destination,
```

```
    // the greeting() method is called.
```

```
    @RequestMapping("/hello")
```

```
    @SendTo("/topic/greetings")
```

```
    // HelloMessage: payload of the message
```

```
    public Greeting greeting>HelloMessage message) throws Exception {
```

```
        Thread.sleep(1000);    // simulated delay
```

```
        return new Greeting("Hello, " + HtmlUtils.htmlEscape(message.getName()) + "!");
```

```
    }
```

```
}
```

# 處理來自客戶端的訊息：無回覆

---

@Controller

```
public class GreetingController {
```

```
    // if a message is sent to the /hello destination,
```

```
    // the greeting() method is called.
```

```
    @RequestMapping("/hello")
```

```
    @SendTo("/topic/greetings")
```

```
    // HelloMessage: payload of the message
```

```
    public void greeting>HelloMessage message) throws Exception {
```

```
        logger.info("無回覆，只處理送來的訊息");
```

```
    }
```

```
}
```

# 前端需要送出的訊息

---

```
function sendData() {  
    if (!name.value || name.value === null  
        || name.value.trim().length == 0) {  
        alert("必須輸入姓名");  
        return;  
    }  
    var dataSent = JSON.stringify({  
        'name' : name.value  
    });  
    stompClient.send("/app/hello", {}, dataSent);  
    console.log("Client: Send Data: " + dataSent);  
}
```

```
public class HelloMessage {  
    String name;  
    public HelloMessage() {  
    }  
    public HelloMessage(String name) {  
        this.name = name;  
    }  
}
```

//以下省略

# 處理來自客戶端的訊息：有回覆

---

```
// if a message is sent to the /hello destination, the greeting() method is called.  
@MessageMapping("/hello")  
@SendTo("/topic/greetings")  
// HelloMessage: payload of the message  
public Greeting greeting>HelloMessage message) throws Exception {  
    Thread.sleep(1000); // simulated delay  
    return new Greeting("Hello, " + HtmlUtils.htmlEscape(message.getName()) + "!");  
}
```

# 後端的回覆訊息：Greeting.java

---

```
public class Greeting {  
    private String content;  
    public Greeting() {  
    }  
    public Greeting(String content) {  
        this.content = content;  
    }  
    public String getContent() {  
        return content;  
    }  
}
```

# 前端程式摘要

---

- 啟用STOMP功能：

```
var socket = new SockJS(contextPath + '/gs-guide-websocket');  
stompClient = Stomp.over(socket);
```

- 訂閱訊息：

```
stompClient.subscribe('/topic/greetings', function (feedback) {  
    showGreeting(JSON.parse(feedback.body).content);  
});
```

```
function connect() {  
    var socket = new SockJS(contextPath + '/gs-guide-websocket');  
    stompClient = Stomp.over(socket);  
    stompClient.connect({}, function (frame) {  
        setConnected(true);  
        console.log('連線成功: ' + frame);  
        stompClient.subscribe('/topic/greetings', function (feedback) {  
            showGreeting(JSON.parse(feedback.body).content);  
        });  
    });  
}
```

# 前端程式摘要

---

- 送出訊息：

```
var dataSent = JSON.stringify({  
    'name' : name.value  
});  
stompClient.send("/app/hello", {}, dataSent);
```

- STOMP send()方法：

```
(void) send(destination, headers = {}, body = "")
```

```
function sendData() {  
    if (!name.value || name.value === null  
        || name.value.trim().length == 0) {  
        alert("必須輸入姓名");  
        return;  
    }  
    var dataSent = JSON.stringify({  
        'name' : name.value  
    });  
    stompClient.send("/app/hello", {}, dataSent);  
    console.log("Client: Send Data: " + dataSent);  
}
```

# 在任意類別中發送訊息給前端程式

- o.s.m.s.SimpMessagingTemplate類別的  
convertAndSend(destination, payload); 可將payload送往  
destination指定的訊息代理。

@Autowired

```
private SimpMessagingTemplate template;
```

@Override

```
public void run() {  
    count++;  
    Date d = new Date();  
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
    template.convertAndSend("/topic/greetings", new Greeting(sdf.format(d)));  
    if (count == 5) {  
        timer.cancel();  
        timer.purge();  
        timer = null;  
        template.convertAndSend("/topic/greetings", new Greeting("計時結束..."));  
        return;  
    }  
}
```



---

## 五、聊天室

參考資料：<https://www.youtube.com/watch?v=U4lqTmFmbAM>

# WebSocketMessageBrokerConfig.java

---

```
package com.infotran.ws.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
@EnableWebSocketMessageBroker
public class WebSocketMessageBrokerConfig implements WebSocketMessageBrokerConfigurer{

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/chat-example").withSockJS();
    }
    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.setApplicationDestinationPrefixes("/app");
        registry.enableSimpleBroker("/topic");
    }
}
```

# ChatController.java

---

```
package com.infotran.ws.controller;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.Payload;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.messaging.simp.SimpMessageHeaderAccessor;
import org.springframework.stereotype.Controller;
import com.infotran.ws.model.ChatMessage;
@Controller
public class ChatController {
    @MessageMapping("/chat.send")
    @SendTo("/topic/public")
    public ChatMessage sendMessage(@Payload ChatMessage chatMessage) {
        return chatMessage;
    }
    @MessageMapping("/chat.newUser")
    @SendTo("/topic/public")
    public ChatMessage newUser(@Payload ChatMessage chatMessage,
                               SimpMessageHeaderAccessor headAccessor
                               ) {
        headAccessor.getSessionAttributes()
            .put("username", chatMessage.getSender());
        return chatMessage;
    }
}
```

# WebSocketEventListener.java

---

```
package com.infotran.ws.controller;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.event.EventListener;
import org.springframework.messaging.simp.SimpMessageSendingOperations;
import org.springframework.messaging.simp.stomp.StompHeaderAccessor;
import org.springframework.stereotype.Component;
import org.springframework.web.socket.messaging.SessionConnectedEvent;
import org.springframework.web.socket.messaging.SessionDisconnectEvent;

import com.infotran.ws.model.ChatMessage;
import com.infotran.ws.model.MessageType;

@Component
public class WebSocketEventListener {

    public static final Logger logger = LoggerFactory.getLogger(WebSocketEventListener.class);
```

# WebSocketEventListener.java

---

```
private SimpMessageSendingOperations sendingOperations;

@EventListener
public void handleWebSocketConnectListener(SessionConnectedEvent event ) {
    logger.info("DingDingDong, we have a newly arrived connection.");
}

public void handleWebSocketDisconnectListener(SessionDisconnectEvent event ) {

    StompHeaderAccessor headerAccessor = StompHeaderAccessor.wrap(event.getMessage());
    String username = (String) headerAccessor.getSessionAttributes().get("username");

    logger.info("DingDingDong, we have a newly arrived connection.");

    ChatMessage chatMessage = ChatMessage.builder()
        .type(MessageType.DISCONNECT)
        .sender(username)
        .build();

    sendingOperations.convertAndSend("/topic/public", chatMessage);
}
```

# ChatMessage.java

---

```
package com.infotran.ws.model;
```

```
import lombok.Builder;
```

```
import lombok.Getter;
```

```
@Getter
```

```
@Builder
```

```
public class ChatMessage {  
    private MessageType type;  
    private String content;  
    private String sender;  
    private String time;
```

```
}
```

# MessageType.java

---

```
package com.infotran.ws.model;
```

```
public enum MessageType {  
    CHAT,  
    CONNECT,  
    DISCONNECT  
}
```





---

## 六、附錄

# Lombok

---

- 是一個可用在Java程式中的類別庫(Class Library)
- 官網：<https://projectlombok.org/>
- 經由簡單的註釋替Java類別自動產生重複、無趣但卻非有不可的程式碼(boilerplate)，如Getter / Setter / 預設建構子 / 有參建構子 / hashCode / equals 。
- 使用Lombok類別庫專案必須加入相關的依賴。
- 開發環境也需安裝外掛，否則編譯器無法得知Lombok自動替Java程式加入的程式碼。

# 有關的依賴

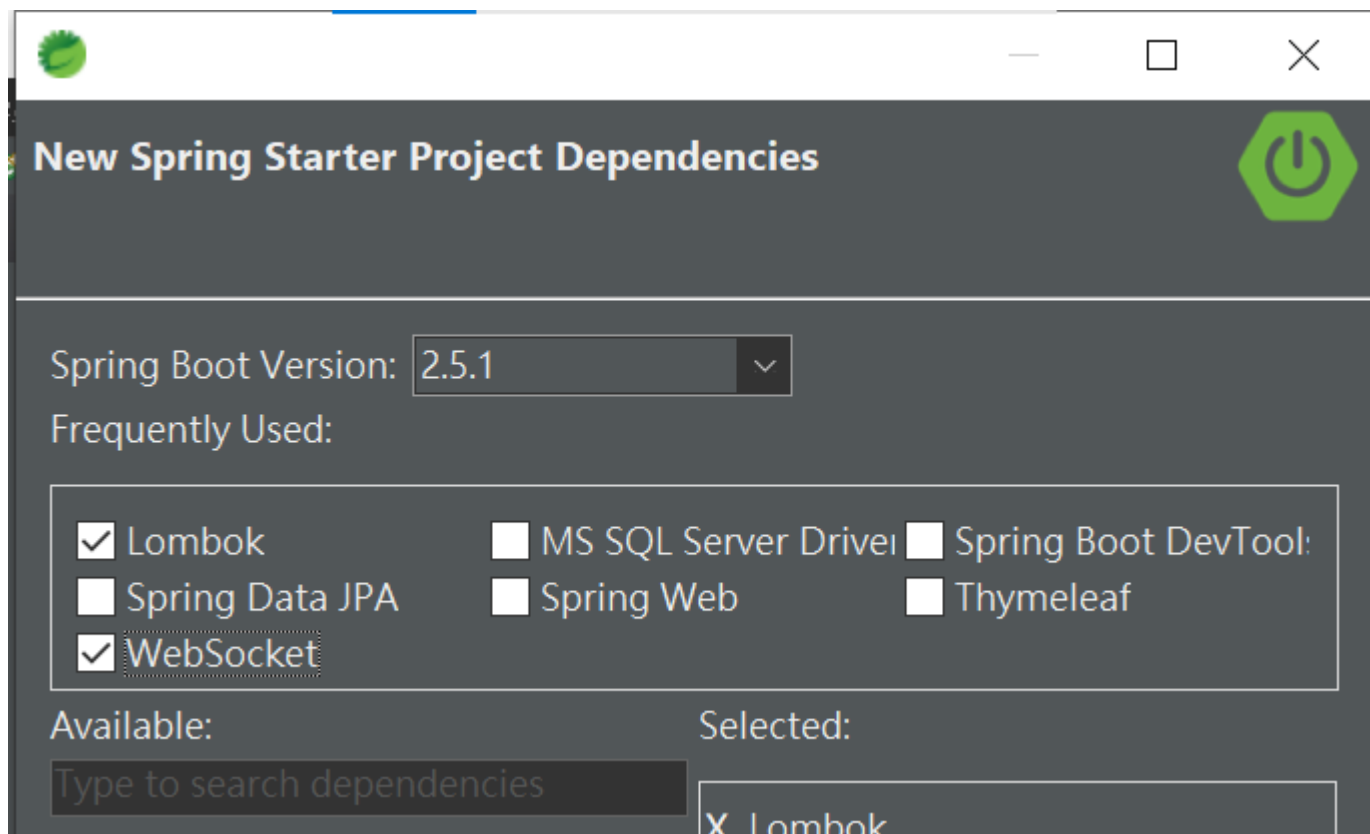
<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

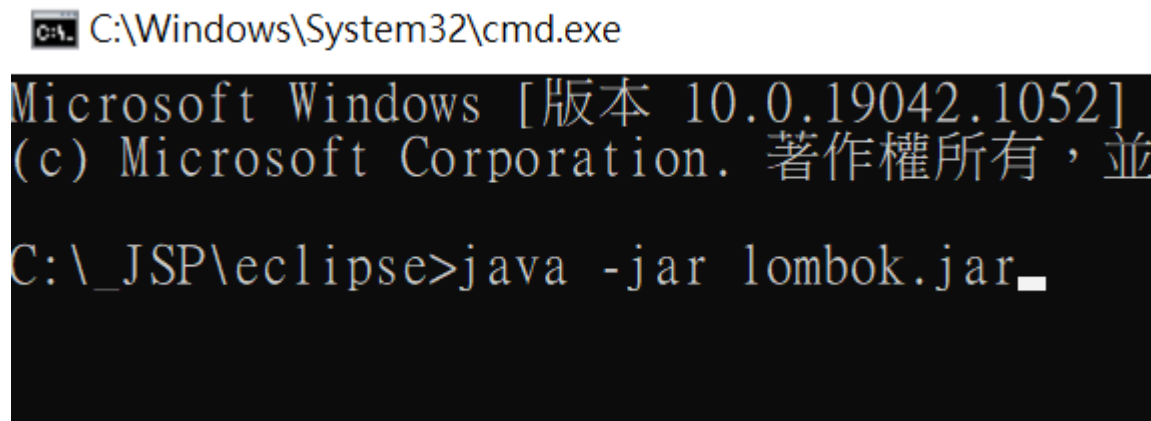
<optional>true</optional>

</dependency>



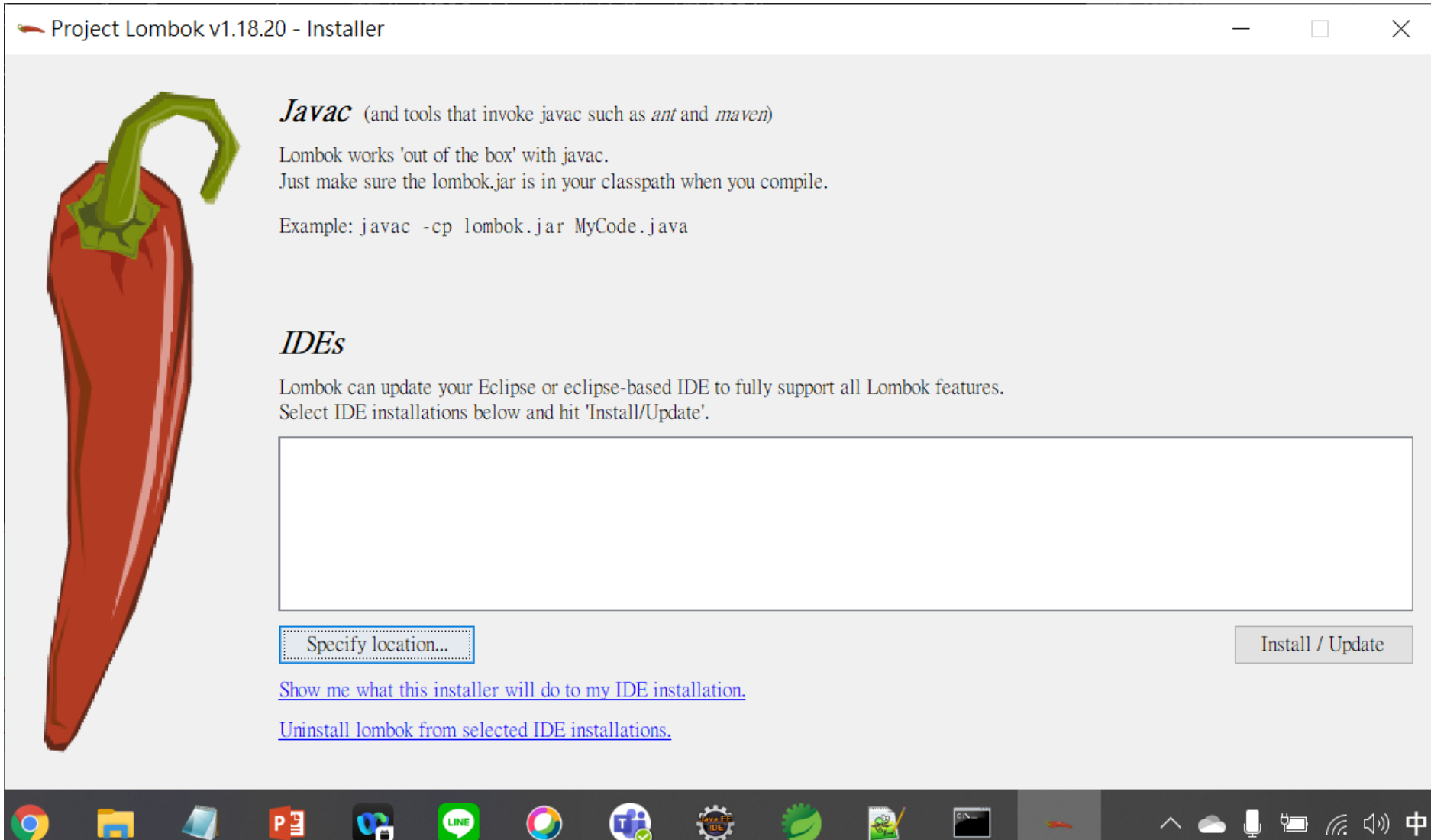
# 替 Eclipse 安裝外掛

- 下載 Lombok Jar 檔
  - <https://projectlombok.org/download>
  - <https://projectlombok.org/downloads/lombok.jar>
- 備份原有的 Eclipse 資料夾
  - C:\\_JSP\eclipse
- 將下載的 lombok.jar 複製到 C:\\_JSP\eclipse
- 進入 DOS 視窗，切換目錄到 C:\\_JSP\eclipse
- 執行



```
C:\Windows\System32\cmd.exe  
Microsoft Windows [版本 10.0.19042.1052]  
(c) Microsoft Corporation. 著作權所有，並  
C:\_JSP\eclipse>java -jar lombok.jar_
```

# lombok



# 找不到Eclipse，必須自行指定Eclipse安裝資料夾

---

Can't find IDE

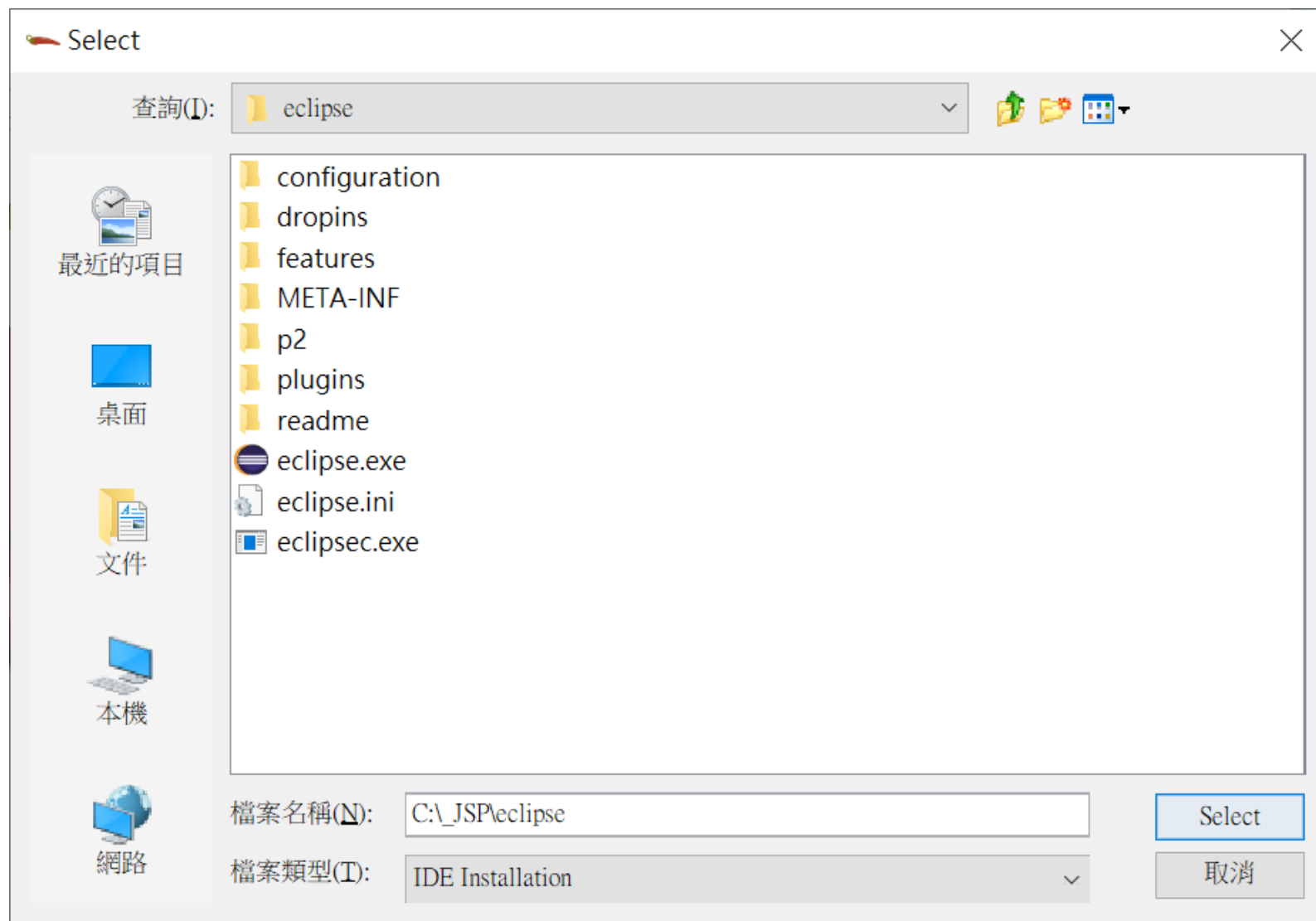


I can't find any IDEs on your computer.

If you have IDEs installed on this computer, please use the 'Specify Location...' button to manually point out the location of your IDE installation to me. Thanks!

確定

# 找到Eclipse安裝資料夾





### *Javac* (and tools that invoke javac such as *ant* and *maven*)

Lombok works 'out of the box' with javac.  
Just make sure the lombok.jar is in your classpath when you compile.

Example: `javac -cp lombok.jar MyCode.java`

### *IDEs*

Lombok can update your Eclipse or eclipse-based IDE to fully support all Lombok features.  
Select IDE installations below and hit 'Install/Update'.

☒ C:\\_JSP\eclipse\eclipse.exe 

[Specify location...](#)

Install / Update

[Show me what this installer will do to my IDE installation.](#)

[Uninstall lombok from selected IDE installations.](#)





## *Install successful*

Lombok has been installed on the selected IDE installations.  
Don't forget to:

- add `lombok.jar` to your projects,
- **exit and start** your IDE,
- **rebuild** all projects!

If you start Eclipse with a custom `-vm` parameter, you'll need to add:  
`-vmargs -javaagent:lombok.jar`  
as parameter as well.

- PLATFORM: JDK16 support added. .
- PLATFORM: All lombok features updated to act in a sane fashion with JDK16's *record* feature. In particular, you can annotate record components with `@NonNull` to have lombok add null checks to your compact constructor (which will be created if need be).
- BUGFIX: Trying to use a lambda expression as parameter to an `@ExtensionMethod` did not work. . (by @Rawi01).
- BUGFIX: `@SuperBuilder` with an existing constructor caused issues in eclipse. . (by @JanRieke).
- BUGFIX: Using `@SuperBuilder` with a handwritten builder class caused issues. . (by @JanRieke).
- BUGFIX: Lombok interacts properly with the new save actions in eclipse 2021-03.
- POTENTIAL BUGFIX: lombok + errorprone could cause `IllegalArgumentException` if using the `MissingSummary` bug pattern. .

# 修改Eclipse.ini

```
14 -vm
15 C:/_JSP/jdk11/bin/javaw.exe
16 -vmargs
17 -Dosgi.requiredJavaVersion=11
18 -Dosgi.instance.area.default=@user.home/ec
19 -XX:+UseG1GC
20 -XX:+UseStringDeduplication
21 --add-modules=ALL-SYSTEM
22 -Dosgi.requiredJavaVersion=11
23 -Dosgi.dataAreaRequiresExplicitInit=true
24 -Xms256m
25 -Xmx2048m
26 --add-modules=ALL-SYSTEM
27 -javaagent:C:\_JSP\eclipse\lombok.jar
28
```

# 可用的註釋

---

- @Getter / @Setter
- @ToString
- @EqualsAndHashCode
- @NoArgsConstructor, @AllArgsConstructor, @RequiredArgsConstructor
- @Data
- @Value
- @Builder
- @Slf4j

# @Getter / @Setter

---

```
package com.example.demo._misc._02.lombok._01;
```

```
import lombok.Getter;
```

```
import lombok.Setter;
```

```
@Getter
```

```
@Setter
```

```
public class Cat {
```

```
String name;
```

```
Double weight;
```

```
}
```

```
package com.infotran.springboot.model;
```

```
public class CatMain {
```

```
    public static void main(String[] args) {
```

```
        Cat c = new Cat();
```

```
        c.setName("Kitty");
```

```
        c.setWeight(5.0);
```

```
        System.out.println(c.getName());
```

```
        System.out.println(c.getWeight());
```

```
    }
```

```
}
```

# @ToString

---

```
package com.example.demo._misc._02.lombok._02;  
import lombok.ToString;
```

```
@ToString  
public class Cat {  
    String name = "史嘉飛";  
    Double weight = 3.5;  
}
```

```
package com.example.demo._misc._02.lombok._02;  
public class CatMain {  
  
    public static void main(String[] args) {  
        Cat c = new Cat();  
        System.out.println(c);  
    }  
}
```

# @EqualsAndHashCode

---

```
package com.example.demo._misc._02.lombok._03;
```

```
import lombok.EqualsAndHashCode;
```

```
@EqualsAndHashCode
```

```
public class Cat {
```

```
    String name;
```

```
    Double weight;
```

```
    public Cat(String name, Double weight) {
```

```
        super();
```

```
        this.name = name;
```

```
        this.weight = weight;
```

```
    }
```

```
}
```

# @EqualsAndHashCode

---

```
package com.example.demo._misc._02.lombok._03;

import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class CatMain {

    public static void main(String[] args) {
        Cat c1 = new Cat("史嘉飛", 3.5);
        Cat c2 = new Cat("史嘉飛", 3.5);
        Cat c3 = new Cat("史嘉飛", 3.5);

        List<Cat> sourceList = Arrays.asList( c1, c2, c3);
        Set<Cat> set = new HashSet<>(sourceList);
        System.out.println("Set之元素: " + set.size());
    }
}
```

# @NoArgsConstructor / @AllArgsConstructor

---

```
package com.example.demo._misc._02.lombok._04;
```

```
import lombok.AllArgsConstructor;  
import lombok.NoArgsConstructor;
```

```
@NoArgsConstructor  
@AllArgsConstructor  
public class Cat {  
    String name;  
    Double weight;  
}
```



# @NoArgsConstructor / @AllArgsConstructor

---

```
package com.example.demo._misc._02.lombok._04;

import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class CatMain {

    public static void main(String[] args) {
        Cat c1 = new Cat("史嘉飛", 3.5);
        Cat c2 = new Cat("史嘉飛", 3.5);
        Cat c3 = new Cat();
        Cat c4 = new Cat();

        List<Cat> sourceList = Arrays.asList(c1, c2, c3, c4);
        Set<Cat> set = new HashSet<>(sourceList);
        System.out.println("Set之元素: " + set.size());
    }
}
```

# @RequiredArgsConstructor

---

- 產生一個包含『特定參數』的 constructor，特定參數指的是那些有加上 **final** 修飾詞的實例變數們

```
package com.example.demo._misc._02.lombok._05;
```

```
import lombok.RequiredArgsConstructor;  
import lombok.ToString;
```

```
@RequiredArgsConstructor  
@ToString  
public class Cat {  
    final String name;  
    Double weight;  
}
```

# @RequiredArgsConstructor

---

```
package com.example.demo._misc._02.lombok._05;
```

```
public class CatMain {  
  
    public static void main(String[] args) {  
        Cat c1 = new Cat("史嘉飛");  
        System.out.println("c1: " + c1);  
    }  
}
```

# @Data

---

- 懶人包註釋，效果等於
  - @Getter/@Setter
  - @ToString
  - @EqualsAndHashCode
  - @RequiredArgsConstructor

# @Data

---

```
package com.example.demo._misc._02.lombok._06;  
import lombok.AllArgsConstructor;  
import lombok.Data;
```

```
@Data  
@AllArgsConstructor  
public class Cat {  
    String name;  
    Double weight;  
}
```

```
package com.example.demo._misc._02.lombok._06;  
public class CatMain {  
  
    public static void main(String[] args) {  
        Cat c1 = new Cat("史嘉飛", 5.0);  
        System.out.println("c1: " + c1);  
    }  
}
```

# @Value

---

- 懶人包註釋，但是它會將所有的實例變數都設成 `final` 的，其他則與 `@Data` 一樣，等於同時加了以下注解

`@Getter` (注意沒有 `setter`)

`@ToString`

`@EqualsAndHashCode`

`@RequiredArgsConstructor`

# @Value

---

```
package com.example.demo._misc._02.lombok._07;  
import lombok.Value;
```

```
@Value  
public class Cat {  
    String name = "Kitty";  
    Double weight = 3.0;  
}
```

```
package com.example.demo._misc._02.lombok._07;  
public class CatMain {  
  
    public static void main(String[] args) {  
        Cat c1 = new Cat();  
        System.out.println("c1: " + c1);  
    }  
}
```

# @Builder

---

```
package com.example.demo._misc._02.lombok._08;
```

```
import lombok.Builder;  
import lombok.ToString;
```

```
@Builder  
@ToString  
public class Cat {  
    String name;  
    Double weight;  
}
```



# @Builder

---

```
package com.example.demo._misc._02.lombok._08;
```

```
public class CatMain {  
  
    public static void main(String[] args) {  
        Cat c1 = Cat.builder()  
            .name("史嘉飛")  
            .weight(5.0)  
            .build();  
  
        System.out.println("c1: " + c1);  
    }  
}
```

- 自動產生該類別的 log 靜態變數，極為方便的使用日誌，不用再定義 log 靜態常數。

```
private static final org.slf4j.Logger log =  
    org.slf4j.LoggerFactory.getLogger(Cat.class);
```

# @Slf4j

---

```
package com.example.demo._misc._02.lombok._09;

import lombok.extern.slf4j.Slf4j;

@Slf4j
public class Cat {
    // private static final org.slf4j.Logger log =
    //      org.slf4j.LoggerFactory.getLogger(Cat.class);
    String name;
    Double weight;
    public Cat(String name, Double weight) {
        super();
        this.name = name;
        this.weight = weight;
        log.info("Cat 建構中");
    }
}
```