WebSocket

段維瀚 老師

WebSocket

- JSR 356 WebSocket 是一種透過 TCP 通訊協定提供全雙工雙向資料交換的通訊技術
  - 並透過 http 建立資料交換需求
  - 系統支援
    - Tomcat 8 開始支援
    - 支援 WebSocket 的瀏覽器（有提供 WebSocket JS API）
      - Google Chrome、Firefox、Safari、Microsoft Edge、Internet Explorer和Opera。

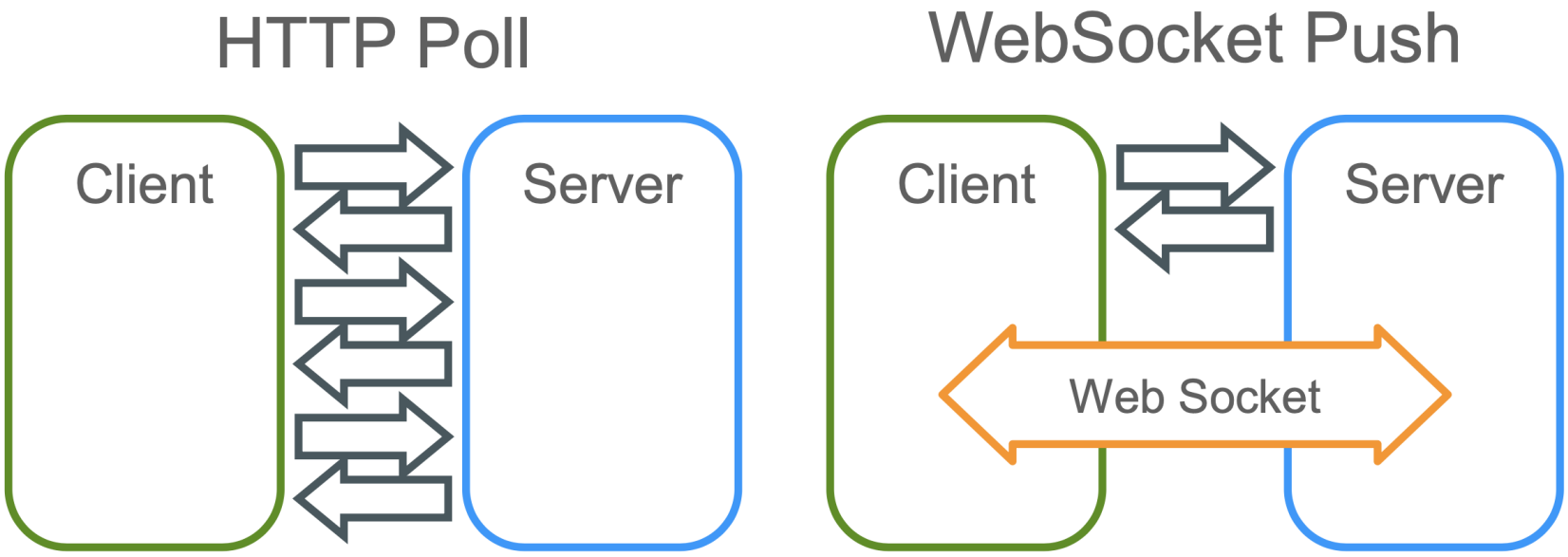
WebSocket 優點

- 優點主要是能夠雙向通信，低延遲和較小的通信開銷。
- WebSocket通常適用於需要發送大量相對較小的消息（如在線遊戲或市場報價廣播）的Web應用程序。
  - WebSocket參考實現
    - Tyrus(泰魯斯)
      https://tyrus-project.github.io/

Why WebSocket

- HTTP 是一種無狀態、短連接、用完即關的通訊協定
- 實務上需要能夠即時傳遞資料且Always在線的技術
  - 股市行情、通訊軟體、即時訊息需要等
- 以往採用輪詢、定向、定時建立連線查詢資料，效能差、無法即時。

Why WebSocket



瀏覽器支援 WebSocket

	IE	Chrome	Safari	Opera	Firefox	Android Browser	BlackBerry Browser
支援	10.0	14.0~29.0	6.0	12.1~16.0	6.0~23.0	X	7.0~10.0
部分支援	X	4.0~13.0	5.0~5.1	11.0~12.0	4.0~5.0	X	X
不支援	5.0~9.0	X	3.1~4.0	9.0~10.6	2.0~3.6	2.1~4.2	X

WebSocket

- WebSocket 應用程式分為
  - Server
    - 發佈 WebSocket Endpoint 提供連線 URIs
      - ws://127.0.0.1:8080/WebSocket/websocket
      - wss://127.0.0.1:8080/WebSocket/websocket
  - Client
    - 透過 Endpoint URIs 向 Server 建立連線需求
  - WebSocket 在完成建立連線後，即已完成雙方對稱狀態，任何一方皆可自由接收與傳遞訊息，並可以任意關閉連線。

WebSocket

- Server端撰寫有二種方法
  - 透過繼承 Endpoint
    - ServerEndpointConfig.Builder.create(xxx.class, "/websocket").build();
  - 使用 @ Annotation
    - @ServerEndpoint("/websocket")
    - @ServerEndpoint("/chatroom/{room-no}")
- 最後得到的 Endpoint URIs
  - ws://127.0.0.1:8080/xxx/websocket
  - ws://127.0.0.1:8080/xxx/chatroom/101

WebSocket

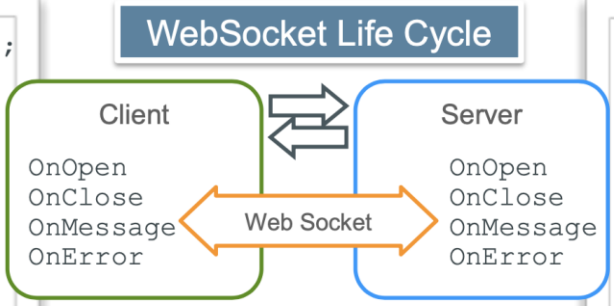
- ws://127.0.0.1:8080/xxx/websocket
  - @OnOpen
    public void onOpen(Session session) {
- ws://127.0.0.1:8080/xxx/chatroom/101
  - @OnOpen
    public void onOpen(Session session,
    @PathParam("room-no") String roomNo) {

WebSocket

- Annotation based model

註解	描述
`@ServerEndpoint`	如果使用修飾 `@ServerEndpoint` ，則該容器可確保該類作為偵聽特定URI空間的WebSocket服務器的可用性
`@ClientEndpoint`	用此註釋裝飾的類被視為WebSocket客戶端
`@OnOpen`	`@OnOpen` 啟動新的WebSocket連接時，對Java方法進行註釋，以供容器調用該方法
`@OnMessage`	將 `@OnMessage` 消息發送到端點時，用註釋的Java方法從WebSocket容器接收
`@OnError`	裝飾一個方法， `@OnError` 以便在WebSocket通信出現問題時調用該方法
`@OnClose`	用於裝飾您要在WebSocket連接關閉時由容器調用的Java方法

```
var socket = new WebSocket("uri");
socket.onopen
    = function(event){...}
socket.onclose
    = function(event){...}
socket.onmessage
    = function(event){...}
socket.onerror
    = function(event){...}
```

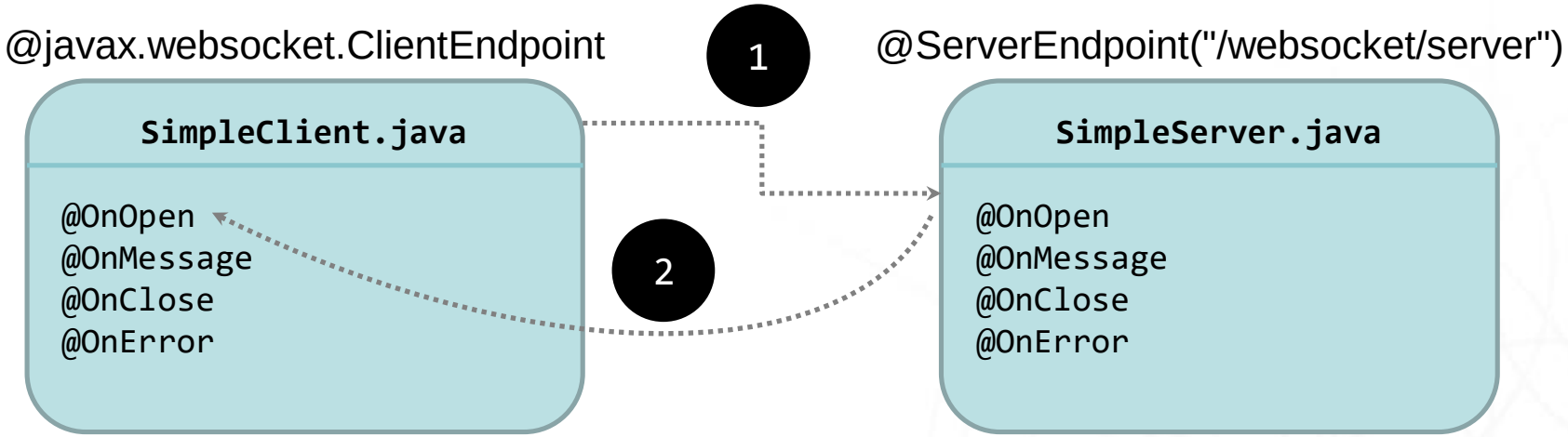



```
@ServerEndpoint("uri")
public class ServerHandler {
    @OnOpen
    public void openSocket...
    @OnClose
    public void closeSocket...
    @OnMessage
    public void handleMessage...
    @OnError
    public void handleError...
}
```

開發前準備與部屬

- Server
  - Tomcat 8 / TomEE 8
- Client
  - Project Tyrus
    - Tyrus是開放源代碼JSR 356-WebSocket參考實現的Java API，可輕鬆開發WebSocket應用程序。
      - pom.xml
        - » <dependency>
          <groupId>org.glassfish.tyrus.bundles</groupId>
          <artifactId>tyrus-standalone-client-jdk</artifactId>
          <version>1.13</version>
          </dependency>

WebSocket 運作



```
@ServerEndpoint("/websockettest")
public class WebSocketEndpointTest {
   // 用來存放WebSocket已連接的Socket
   static CopyOnWriteArraySet<Session> sessions;
   @OnMessage
   public void onMessage(String message, Session session) throws IOException,
         InterruptedException, EncodeException {
   }
   @OnOpen
   public void onOpen(Session session) {
      //紀錄連接到sessions中
      if (sessions == null) {
         sessions = new CopyOnWriteArraySet<Session>();
      }
      sessions.add(session);
   }
   @OnClose
   public void onClose(Session session) {
      //將連接從sessions中移除
      if (sessions == null) {
         sessions = new CopyOnWriteArraySet<Session>();
      }
      sessions.remove(session); }
}
```

WebSocket Server 撰寫套版（Endpoint）

CopyOnWrite

- CopyOnWrite 容器即寫時複製的容器。
  - 一般的理解是當我們往一個容器添加元素的時候，不直接往當前容器添加，而是先將當前容器進行Copy，複製出一個新的容器，然後新的容器裡添加元素，添加完元素之後，再將原容器的引用指向新的容器。這樣做的好處是我們可以對CopyOnWrite容器進行並發的讀，而不需要加鎖，因為當前容器不會添加任何元素。所以CopyOnWrite容器也是一種讀寫分離的思想，讀和寫不同的容器。

WebSocket

- Server 端收送訊息
  - .getBasicRemote().sendText();
    - 同步 / blocking
  - .getAsyncRemote().sendText();
    - 非同步 / non-blocking

```
getSendTimeout()                                  long
sendBinary(ByteBuffer arg0)                       Future<Void>
sendBinary(ByteBuffer arg0, SendHandler arg1)     void
sendObject(Object arg0)                           Future<Void>
sendObject(Object arg0, SendHandler arg1)         void
sendPing(ByteBuffer applicationData)              void
sendPong(ByteBuffer applicationData)              void
sendText(String arg0)                             Future<Void>
sendText(String arg0, SendHandler arg1)           void
setBatchingAllowed(boolean allowed)               void
setSendTimeout(long arg0)                         void
```

WebSocket

```
// 用來存放WebSocket已連接的Socket
static CopyOnWriteArraySet<Session> sessions;
```

接收

也可替換成
session.getOpenSessions()

```
@OnMessage
public void onMessage(String message, Session session) throws IOException,
        InterruptedException, EncodeException {
    for (Session s : sessions) {    //對每個連接的Client傳送訊息
        if (s.isOpen()) {
            s.getAsyncRemote().sendText(message);
        }
    }
}
```

傳送

```
// 設置WebSocket
function setWebSocket() {
   var url =  'ws://127.0.0.1:8080/WebSocket/websocket' ;
   // 開始WebSocket連線
   webSocket = new WebSocket(url);
   // 以下開始偵測WebSocket的各種事件
   //onerror , 連線錯誤時觸發
   webSocket.onerror = function (event) {
      …
   };
   // onopen , 連線成功時觸發
   webSocket.onopen = function (event) {
     …
     webSocket.send(xxx));
   };
   // onmessage , 接收到來自Server的訊息時觸發
   webSocket.onmessage = function (event) {
   …
   };
}
```

WebSocket Client 撰寫套版（Javascript）

```
// 傳送訊息
webSocket.send(xxx)
```

WebSocket JS API → https://developer.mozilla.org/en-US/docs/Web/API/WebSocket/WebSocket


JUnit
JS
ORACLE
FUSION MIDDLEWARE
WEBLOGIC SERVER
JBoss
by Red Hat
HIBERNATE
HTML
ORACLE
SPATIAL
PostGIS
SoapUI
by SMARTBEAR
spring
by Pivotal
GeoServer
CSS
eclipse link
OpenLayers
SUBVERSION
jQuery
write less, do more.
PRIME FACES
Java™
Microsoft
SQL Server
mongoDB
NetBeans
maven
git
MySQL
ORACLE
DATABASE
ANGULARJS
by Google
LIFERAY
eclipse
PostgreSQL
sonarqube
Visual Studio