Spring Web Socket

강병준

Spring Web Socket

1. 웹 서버 소켓을 일반 Java Web Project의 형태로 구현하게 되면 DispatcherServlet의 연동이나 스프링 빈 객체를 사용하는 것이 번거롭습니다.
2. 스프링에서는 웹 소켓 서버를 구현하는데 필요한 인터페이스인 WebSocketHandler를 지원하고 있습니다.
3. Spring4의 웹 소켓은 서블릿 3.0이상에서만 동작합니다.
4. 웹 소켓 서버를 구현하기 위한 작업
   1) WebSocketHandler 인터페이스를 구현한 클래스를 생성
   2) DispatcherServlet.xml 파일에 <websocket:handlers> 또는 @EnableWebSocket 어노테이션을 이용해서 앞에서 구현한 클래스의 객체를 웹소켓 앤드포인트로 등록

Spring Web Socket

1. WebSocketHandler 인터페이스
   1) void afterConnectionClosed(WebSocketSession session, CloseStatus closeStatus): 소켓 연결이 종료 된 후 호출되는 메소드
   2) void afterConnectionEstablished(WebSocketSession session): 연결이 되고 호출되는 메소드
   3) void handleMessage(WebSocketSession session, WebSocketMessage<?> message) : 메시지를 받았을 때 호출되는 메소드
   4) void handleTransportError(WebSocketSession session, Throwable exception): 전송 중 에러가 발생했을 때 호출되는 메소드
   5) boolean supportsPartialMessages():: 대량의 데이터를 나누어 받을 것인지 여부를 설정하는 메소드

2. 인터페이스를 implements 한 클래스
   1) TextWebSocketHandler
   2) AbstractWebSocketHandler

Spring Web Socket

1. WebSocketMessage 인터페이스
   1) T getPayLoad(): 실제 메시지
   2) Boolean isLast(): 마지막 인지의 여부

2. WebSocketSession
   1) String getId()
   2) URI getUri()
   3) InetSocketAddress getLocalAddress()
   4) InetSocketAddress getRemoteAddress()
   5) boolean isOpen()
   6) sendMessage(WebSocketMessage)
   7) void close()

Spring Web Socket

1. Spring 프로젝트를 생성하고 pom.xml 파일에서 스프링 버전(4.0.0 이상) 수정
2. <org.springframework-version>**4.2.4.RELEASE**</org.springframework-version>
3. pom.xml 파일에서 servlet과 jsp 버전 변경

```
<!-- Servlet -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>javax.servlet-api</artifactId>
			<version>3.0.1</version>
		</dependency>
		<dependency>
			<groupId>javax.servlet.jsp</groupId>
			<artifactId>jsp-api</artifactId>
			<version>2.2</version>
		</dependency>
```

Spring Web Socket

web.xml 파일의 dtd 변경

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">
```

pom.xml 파일에서 websocket추가

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-websocket</artifactId>
     <version>${org.springframework-version}</version>
</dependency>
```

Spring Web Socket

Spring Web Socket 서버를 위한 TextWebSocketHandler를 상속받은 클래스를 생성

```
import java.util.*;
import org.springframework.web.socket.*;
import org.springframework.web.socket.handler.TextWebSocketHandler;
public class ChatWebSocketHandler extends TextWebSocketHandler{
    // 접속한 클라이언트들의 Session을 저장 할 객체 생성
    Map<String,WebSocketSession>users =
            new HashMap<String,WebSocketSession>();
    // 클라이언트가 연결될 때 호출되는 메소드
    // 클라이언트를 Map에 저장
    public void afterConnectionEstablished(WebSocketSession session){
            users.put(session.getId(), session);
    }
    // 클라이언트의 연결이 해제될 때 호출되는 메소드
    // Map에서 제거
    public void afterConnectionClosed(WebSocketSession session,
                        CloseStatus status){
          users.remove(session.getId());
    }
```

Spring Web Socket

```
    // 클라이언트에서 메시지가 왔을 때 호출되는 메소드
    // 메시지를 모든 클라이언트에게 전송
    public void handleTextMessage(WebSocketSession session,
                        TextMessage message) throws Exception{
        // 전송되어 온 메시지
        // String msg=message.getPayload();
        // 앞의 4글자를 제외한 부분을 가지고 메시지 만들기
        // TextMessage mes = new TextMessage(msg.substring(4));
        // Map의 모든 Value를 가져오기
        Collection<WebSocketSession> set=users.values();
        // set의 모든 구성 요소에 mes를 전송
        for(WebSocketSession s : set){
            s.sendMessage(mes);
        }
    }
}
```

Spring Web Socket

servlet-context.xml 파일에 디폴트 서블릿 설정과 WebSocket 객체를 주소와 바인딩하는 코드를 작성

```
<!-- 이 코드는 거의 필수 -->
<default-servlet-handler/>
<!-- 웹 소켓 등록 -->
<!-- 웹 소켓 핸들러의 빈을 생성 -->
<beans:bean id="chatHandler“
        class="webtest.service.ChatWebSocketHandler"/>
<websocket:handlers>
    <websocket:mapping handler="chatHandler" path="/chat-ws.do">
    </websocket:mapping>
</websocket:handlers>
```

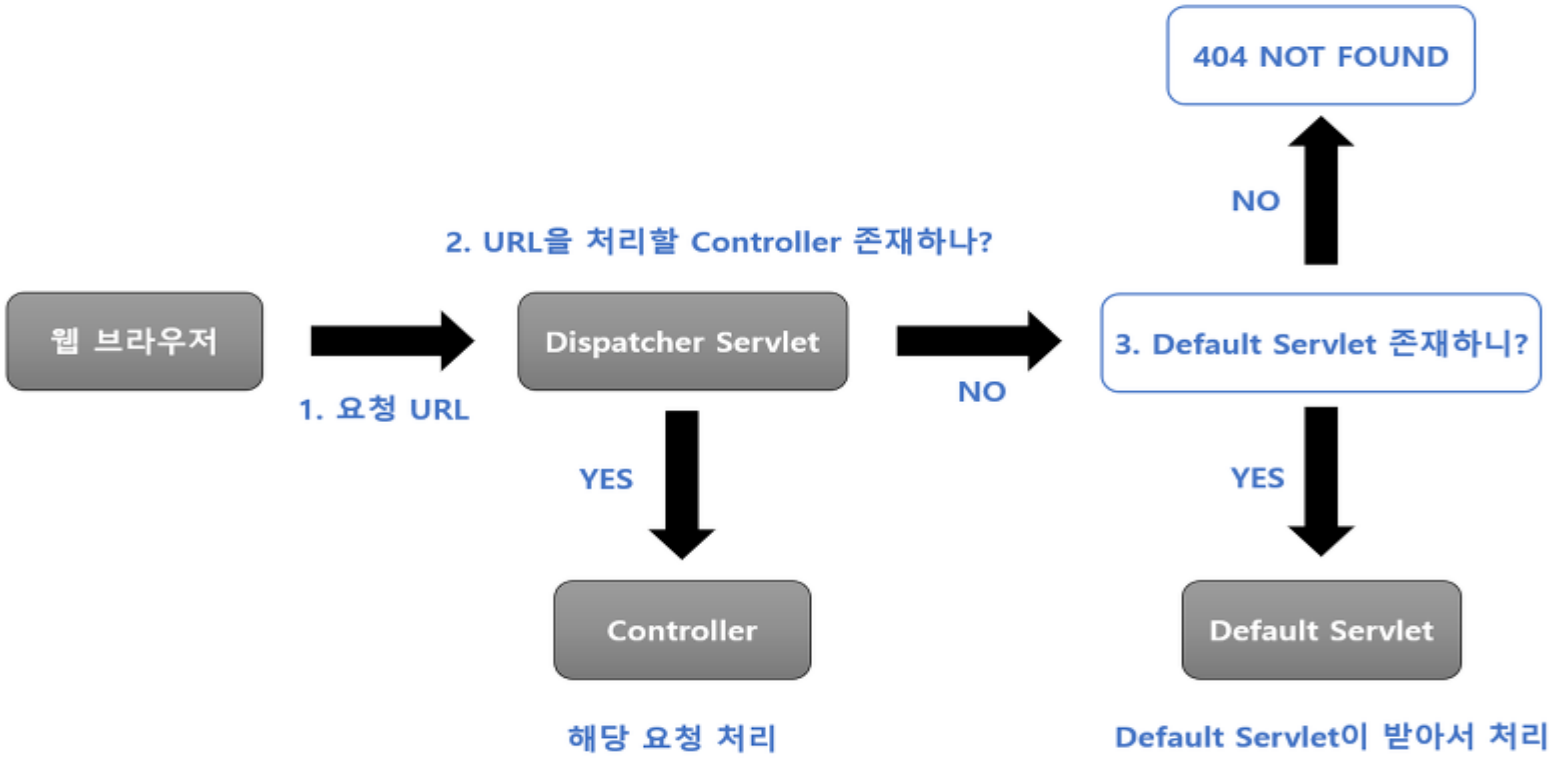
<MVC :default-servlet-handler/> 개념

<MVC :default-servlet-handler/>

: DispatcherSerlvet이 **처리하지 못한 요청**을 **DefaultSerlvet에게 넘겨주는 역할**을 하는 핸들러

*.css와 같은 컨트롤러에 매핑되어 있지 않은 URL 요청은 최종적으로 Default Servlet에 전달되어 처리하는 역할

동작 순서



<MVC :default-servlet-handler/> 개념

-쓰는 이유

DispatcherServlet의 매핑이 "/"로 지정하면 JSP를 제외한 모든 요청이 DispatcherServlet으로 가기 때문에,

WAS가 제공하는 Default Servlet이 *.html, *.css같은 요청을 처리할 수 없게됨.

Default ServletHandler는 이런 요청들을 Default Servlet에게 전달해주는 Handler이다.

요청 URL에 매핑되는 컨트롤러가 존재하지 않을 때, 404응답 대신,

DefaultServlet이 해당 요청 URL을 처리하도록 함.

-사용 방법

```
<web.xml>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

<MVC :default-servlet-handler/> 개념

<servlet-context.xml>

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


<mvc:default-servlet-handler />
```

/<mvc:default-servelt-handler/>를 선언하면, Default ServletHandler가 Bean으로 등록되며 동작함.

* <mvc:default-servlet-handler/>는 Dispatcherservlet 매핑을 "/"로 지정할 때 다른 요청들을 위해 써야한다.

event.keycode

event.keycode
-> 키보드 눌렀을 때 그 키에 대해 이벤트를 발생시키고 싶을 때 사용

※ 브라우저에 따라 이벤트를 감지하는 방식이 다름
- IE계열(internet explorer) : window.event
- 비IE계열(크롬, 파이어폭스 등) : event

※ 브라우저에 따라 문자코드를 받는 방식이 다름
- IE계열(internet explorer) : event.keycode
- 비IE계열(크롬, 파이어폭스 등) : event.which

사용예)

```
<form name="SearchForm" method="post" action="XXX.html">
<input name="searchKeyword" type="text" onkeypress='fn_HandleEnter_top(event,
document.SearchForm);'/></form>
function fn_HandleEnter_top (event, thisform) {
    var keyCode=event.keyCode?event.keyCode:event.which?event.which:event.charCode;
    if (keyCode == 13) {   document.SearchForm.submit();  return false;
    } else {   return true;  }
}
```

Spring Web Socket

해당서버 IP로 시작해야 작동됨

index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
        <script type="text/javascript">
            location.href="http://192.168.0.10:8181/ch11/chat.do";
        </script>
</body>
</html>
```

Spring Web Socket

```
<%@ include file="header.jsp" %>
<style type="text/css">
        table {    height: 450px; border: 2px solid green;
                    table-layout: fixed; overflow: hidden; }
        #chatMessage { height: 400px;  overflow: scroll; }
</style>
<script type="text/javascript">
    var websocket; // 웹 소켓 변수
     $(function() {
      // 전송과 입장 및 퇴장 버튼을 눌렀을 때 메소드를 호출하도록 설정
        $('#enterBtn').click(function() {      connect();        });
        $('#exitBtn').click(function() {       disconnect();     });
        $('#sendBtn').click(function() {       send();           });
         // message라는 id를 가진 영역에서 Enter를 치면 send라는 메소드 호출
        $('#message').keypress(function(event) {
                    //  누른 키보드 값 가져오기
                    var keycode = event.keyCode?event.keyCode:event.which;
                    if (keycode==13) send(); //  누른 키가 엔터이면 send()호출
                    event.stopPropagation();   //  이벤트 전달을 하지 않음
        });
     });
```

Spring Web Socket

```
function connect() {
        websocket =
        new WebSocket("ws://127.0.0.1:8080/ch12/chat-ws.do");
        websocket.onopen = onOpen;
        websocket.onmessage = onMessage;
        websocket.onclose = onClose;
}
 function onOpen(){   // 연결이 되었을 때 호출될 메소드
        var nickname = $('#nickname').val();
        appendMessage(nickname+"님이 입장했습니다");
}
function onMessage(event) {
        var msg = event.data;
        appendMessage(msg);
}
function appendMessage(msg) {
        $('#chatMessage').append(msg+'<br>');
        var objDiv = document.getElementById("chatMessage");
        objDiv.scrollTop = objDiv.scrollHeight;
}
```

Spring Web Socket

```
        function onClose() {//  연결이 해제될 때 호출되는 메소드
                var nickname = $('#nickname').val();
                appendMessage(nickname+"님이 퇴장했습니다");
        }
        function disconnect() {
                websocket.close();
        }
        //  전송 버튼을 눌렀을 때 호출되는 메소드
         function send(){
        var nickname=$('#nickname').val();
                var msg=$('#message').val();
                websock.send(nickname+“=>"+msg);
                $('#message').val("");
         }
</script>
</head>
<body>
```

Spring Web Socket

```
<div class="container">
<table class="table table-hover">
        <tr><td>별명</td><td><input type="text" id="nickname">
                <input type="button" id="enterBtn" value="입장"
                        class="btn btn-info">
                <input type="button" id="exitBtn" value="퇴장"
                class="btn btn-warning"></td></tr>
        <tr><td>메세지</td><td><input type="text" id="message">
                <input type="button" id="sendBtn" value="전송"
                class="btn btn-success"></td></tr>
        <tr><td>대화 영역</td><td>
                <div id="chatMessage"></div></td></tr>
</table>
</div>
```

Spring Web Socket

HomeController에 요청이 왔을 때 페이지로 이동하도록 설정

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class ChatController {
	@RequestMapping("chat")
	public String chat() {
		return "chat";
	}
}
```