**语音功能使用**

# 一、WebSocket使用

## 1.1、前端调用方法

function conn(){

var ws = new WebSocket('ws://localhost:9999/websocket/1');

// 连接成功时:

ws.addEventListener('open', function (event) {

console.log('websocket connected.连接成功');

});

// 收到消息时:

ws.addEventListener('message', function (event) {

var msg = event.data;

console.log("页面接收到消息："+msg);

});

// 连接关闭时:

ws.addEventListener('close', function () {

console.log('websocket closed.');

});

// 绑定到全局变量:

window.chatWs = ws;

}

## 1.2、SpringBoot后端调用WebSocket方法

### 1、引入依赖

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-websocket</artifactId>  
 </dependency>

<dependency>

<groupId>cn.hutool</groupId>

<artifactId>hutool-all</artifactId>

<version>5.8.9</version>

</dependency>

### 2、编写WebSocket配置文件

@Configuration  
public class WebSocketConfig {  
 */\*\*  
 \* 注入ServerEndpointExporter，  
 \* 这个bean会自动注册使用了@ServerEndpoint注解声明的Websocket endpoint  
 \*/* @Bean  
 public ServerEndpointExporter serverEndpointExporter() {  
 return new ServerEndpointExporter();  
 }  
}

### 3、编写WebScoket主类

package com.example.demo1\_3.server;  
  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
import org.springframework.stereotype.Component;  
import cn.hutool.core.util.StrUtil;  
import javax.websocket.\*;  
import javax.websocket.server.PathParam;  
import javax.websocket.server.ServerEndpoint;  
import java.io.IOException;  
import java.util.concurrent.ConcurrentHashMap;  
  
*/\*\*  
 \* @author Liby  
 \* @date 2022-04-25 16:21  
 \* @description:  
 \* @version:  
 \* 访问地址： ws://localhost:9999/websocket/1  
 \*/*@ServerEndpoint(value = "/websocket/{userId}")  
@Component  
public class WebSocket {  
 private final static Logger *logger* = LogManager.*getLogger*(WebSocket.class);  
  
 */\*\*  
 \* 静态变量，用来记录当前在线连接数。应该把它设计成线程安全的  
 \*/* private static int *onlineCount* = 0;  
  
 */\*\*  
 \* concurrent包的线程安全Map，用来存放每个客户端对应的MyWebSocket对象  
 \*/* private static ConcurrentHashMap<String, WebSocket> *webSocketMap* = new ConcurrentHashMap<>();  
  
 */\*\*  
 \* 与某个客户端的连接会话，需要通过它来给客户端发送数据  
 \*/* private static Session *session*;  
 private String userId;  
  
  
 */\*\*  
 \* 连接建立成功调用的方法  
 \*/* @OnOpen  
 public void onOpen(Session session, @PathParam("userId") String userId) {  
 this.*session* = session;  
 System.*out*.println(session+"是否为null");  
 this.userId = userId;  
 *//加入map  
 webSocketMap*.put(userId, this);  
 *addOnlineCount*(); *//在线数加1  
 logger*.info("用户{}连接成功,当前在线人数为{}", userId, *getOnlineCount*());  
 try {  
 sendMessage("服务端发送的信息");  
 } catch (IOException e) {  
 *logger*.error("IO异常");  
 }  
 }  
  
  
 */\*\*  
 \* 连接关闭调用的方法  
 \*/* @OnClose  
 public void onClose() {  
 *//从map中删除  
 webSocketMap*.remove(userId);  
 *subOnlineCount*(); *//在线数减1  
 logger*.info("用户{}关闭连接！当前在线人数为{}", userId, *getOnlineCount*());  
 }  
  
 */\*\*  
 \* 收到客户端消息后调用的方法  
 \*  
 \* @param message 客户端发送过来的消息  
 \*/* @OnMessage  
 public void onMessage(String message, Session session) {  
 *logger*.info("来自客户端用户：{} 消息:{}",userId, message);  
  
 *//群发消息  
 /\*for (String item : webSocketMap.keySet()) {  
 try {  
 webSocketMap.get(item).sendMessage(message);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }\*/* }  
  
 */\*\*  
 \* 发生错误时调用  
 \*  
 \* @OnError  
 \*/* @OnError  
 public void onError(Session session, Throwable error) {  
 *logger*.error("用户错误:" + this.userId + ",原因:" + error.getMessage());  
 error.printStackTrace();  
 }  
  
 */\*\*  
 \* 向客户端发送消息  
 \*/* public void sendMessage(String message) throws IOException {  
 System.*out*.println("session 为 null 了吗？" + this.*session*);  
 this.*session*.getBasicRemote().sendText(message);  
*// this.session.getAsyncRemote().sendText(message);* }  
  
 */\*\*  
 \* 通过userId向客户端发送消息  
 \*/* public void sendMessageByUserId(String userId, String message) throws IOException {  
 *logger*.info("服务端发送消息到{},消息：{}",userId,message);  
 if(StrUtil.*isNotBlank*(userId)&&*webSocketMap*.containsKey(userId)){  
 *webSocketMap*.get(userId).sendMessage(message);  
 }else{  
 *logger*.error("用户{}不在线",userId);  
 }  
  
 }  
  
 */\*\*  
 \* 群发自定义消息  
 \*/* public static void sendInfo(String message) throws IOException {  
 for (String item : *webSocketMap*.keySet()) {  
 try {  
 *webSocketMap*.get(item).sendMessage(message);  
 } catch (IOException e) {  
 continue;  
 }  
 }  
 }  
  
 public static synchronized int getOnlineCount() {  
 return *onlineCount*;  
 }  
  
 public static synchronized void addOnlineCount() {  
 WebSocket.*onlineCount*++;  
 }  
  
 public static synchronized void subOnlineCount() {  
 WebSocket.*onlineCount*--;  
 }  
  
}

### 4、使用WebSocket发送消息给前端

@RestController  
public class MyTestController {  
  
 @Autowired  
 private WebSocket webSocket;  
  
 @RequestMapping("/test\_conn")  
 public void test\_conn(){  
 *//要从后端页面发送消息* try {  
 Thread.*sleep*(3000);  
 *//给前端发送消息* webSocket.sendMessage("客户端你好，这是服务端发送的消息");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
}

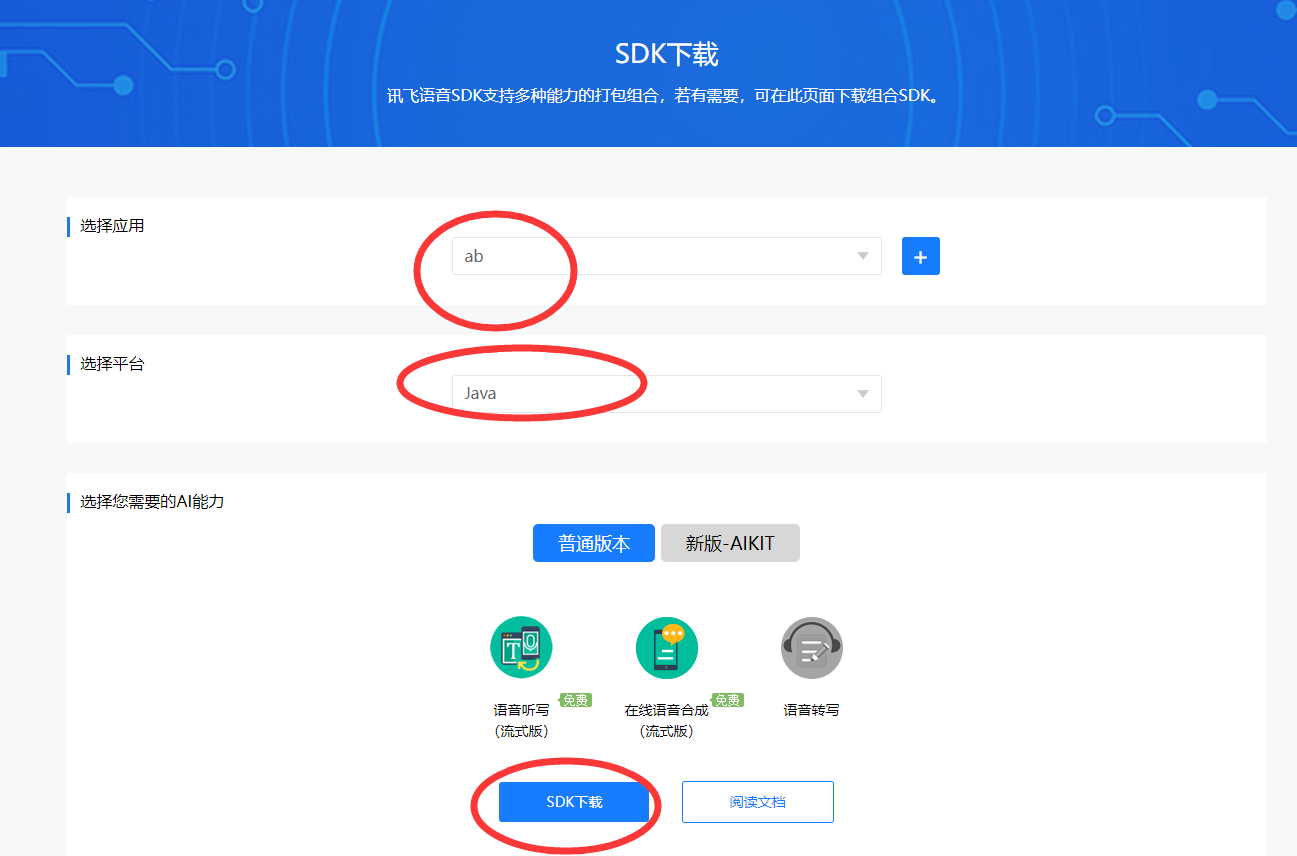
# 二、调用讯飞语音接口

## 2.1、注册讯飞开放平台账号并登陆

## 2.2、控制台---创建应用



## 2.3、下载sdk选择平台，及功能

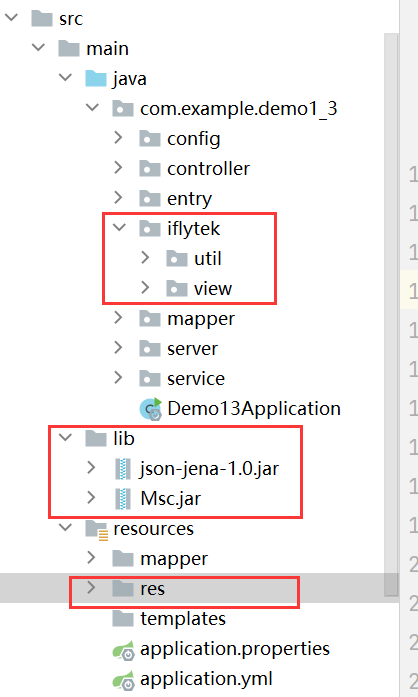


## 2.4、把下载的demo运行

在idea中打开eclips项目

# 三、语音接口整合到SpringBoot项目中

## 3.1、目录结构

# 四、在SpringBoot中如何启动语音平台

多线程的使用

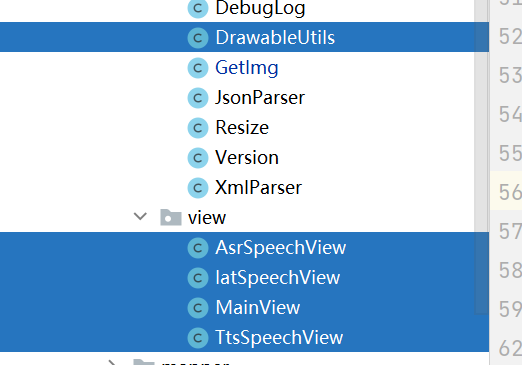
## 1、编写配置类AsyncConfig

package com.example.demo1\_3.config;  
  
import org.springframework.aop.interceptor.AsyncUncaughtExceptionHandler;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.scheduling.annotation.AsyncConfigurer;  
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;  
  
import java.util.concurrent.Executor;  
  
@Configuration  
public class AsyncConfig implements AsyncConfigurer {  
  
 *// ThredPoolTaskExcutor的处理流程  
 // 当池子大小小于corePoolSize，就新建线程，并处理请求  
 // 当池子大小等于corePoolSize，把请求放入workQueue中，池子里的空闲线程就去workQueue中取任务并处理  
 // 当workQueue放不下任务时，就新建线程入池，并处理请求，如果池子大小撑到了maximumPoolSize，就用RejectedExecutionHandler来做拒绝处理  
 // 当池子的线程数大于corePoolSize时，多余的线程会等待keepAliveTime长时间，如果无请求可处理就自行销毁* @Override  
 @Bean  
 public Executor getAsyncExecutor() {  
 ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();  
 *// 核心线程数：线程池创建的时候初始化的线程数* executor.setCorePoolSize(10);  
 *// 最大线程数：线程池最大的线程数，只有缓冲队列满了之后才会申请超过核心线程数的线程* executor.setMaxPoolSize(100);  
 *// 缓冲队列：用来缓冲执行任务的队列* executor.setQueueCapacity(50);  
 *// 线程池关闭：等待所有任务都完成再关闭* executor.setWaitForTasksToCompleteOnShutdown(true);  
 *// 等待时间：等待5秒后强制停止* executor.setAwaitTerminationSeconds(5);  
 *// 允许空闲时间：超过核心线程之外的线程到达60秒后会被销毁* executor.setKeepAliveSeconds(60);  
 *// 线程名称前缀* executor.setThreadNamePrefix("learn-Async-");  
 *// 初始化线程* executor.initialize();  
 return executor;  
 }  
  
 @Override  
 public AsyncUncaughtExceptionHandler getAsyncUncaughtExceptionHandler() {  
 return null;  
 }  
}

## 2、启动多线程



## 3、修改图片读取方式



这五个文件中的图片读取方式需要修改，使用如下工具类读取图片

**GetImg工具类**：

package com.example.demo1\_3.iflytek.util;  
  
import cn.hutool.core.io.resource.ClassPathResource;  
  
import javax.swing.\*;  
import java.io.InputStream;  
  
public class GetImg {  
  
  
 public static ImageIcon getImgIcon(String imgName){  
 ClassPathResource classPathResource = new ClassPathResource(imgName);  
 InputStream is = classPathResource.getStream();  
 ImageIcon imgIcon = null;  
 byte b[] = new byte[2048\*1024];  
 try {  
 is.read(b);  
 }catch (Exception e){  
 e.printStackTrace();  
 imgIcon = new ImageIcon(imgName);  
 }  
 imgIcon = new ImageIcon(b);  
 return imgIcon;  
 }  
}

## 4、创建测试按钮



Service层实现方式：

