实例分享–语音和自然语言控制智能家居

ZigBee作为一种短距离、低功耗的无线通信局域网协议，其优点是超低功耗、安全性高和自组网，并且可容纳多个设备，因此在智能家居控制中占有很大的优势。

同时，随着人工智能、语音识别、自然语义理解的发展，语音控制智能家居将成为一种趋势，这里会以window java应用程序为例，讲解如何通过语音识别控制智能家居，并输出ZigBee3.0协议，也很方便和ZigBee协调器进行对接，实现语音直接控制硬件。

下面详细介绍程序的功能和代码实现，希望语音、语义理解今后能广泛的应用在家居等控制领域。

代码下载

语音和自然语言控制智能家居输出Zibee3.0协议实例源码

注：下载代码后请仔细阅读说明文档。

APP 测试请查看第3节。

功能分析

APP 工作流程

APP的工作流程如下图所示，图中虚线框部分均由OLAMI开发平台提供，后面会具体介绍OLAMI开发平台的使用方法。

其余部分由APP来完成

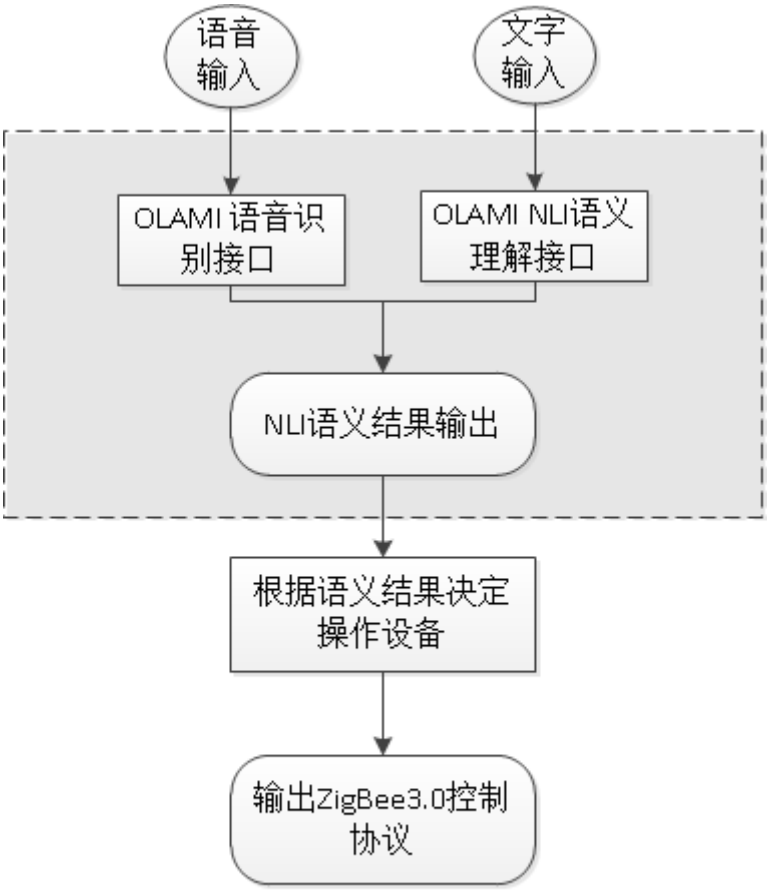


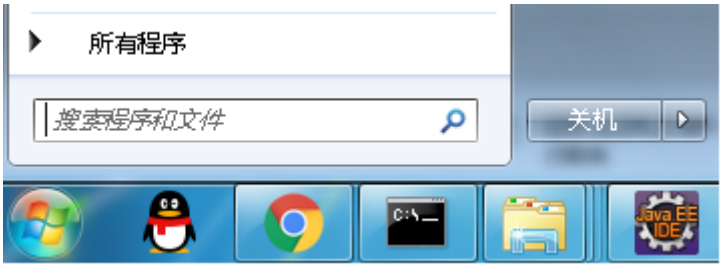
Fig. APP 工作流程图

语音输入

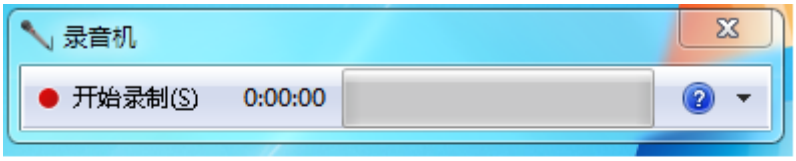
OLAMI的语音识别支持两种格式：

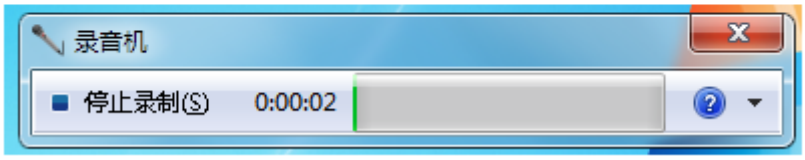
> WAV 格式的 PCM 录音数据，单声道（mono）、16K 采样率（16 KHz Sample Rate）、16 bits 位深（Bit Resolution）。
>
> Speex 音频压缩，节省数据传输量，压缩参数：Wideband 模式、Quality（压缩比）= 10、单声道（mono）、16K 采样率（16 KHz Sample Rate）。

首先要确保硬件设备没有问题，可以进行正常的语音录入。在电脑上安装好麦克风之后，在“开始菜单”中输入“录音机”。



然后在弹出的录音机中点击“开始录音”，使用话筒录音后点击“停止录音”后会弹出保存录音结果的对话框，保存，听听声音正常即可。当然，也可以使用QQ等第三方测试麦克风的软件。





确定硬件设备无误之后，只要通过javax.sound.sampled.TargetDataLine调用windows录音功能,录下符合OLAMI语音识别接口的声音数据即可，我的录音方式是一边录音，同时将原始数据通过speex压缩的方式post给 OLAMI 语音识别的API接口。不是保存为wav文件之后再上传，这样能够提高语音识别的效率。

文字输入

文字输入即直接文本输入，比如“打开空调”，“把彩灯调成红色”。

处理NLI输出

即根据OLAMI NLI的语义输出结果决定如何操作设备，比如当输入为“打开灯”时，我们可以收到如下JSON数据：


```
{
    "data": {
        "asr": {
            "result": "打开灯",
            "speech_status": 0,
            "final": true,
            "status": 0
        },
        "seg": "打开 灯 ",
        "nli": [
            {
                "desc_obj": {
                    "status": 0
                },
                "semantic": [
                    {
                        "app": "smarthome",
                        "input": "打开灯",
                        "slots": [
                            {
```

```
                                    "modifier": [
                                        "open"
                                    ],
                                    "name": "control_obj",
                                    "value": "灯"
                                }
                            ],
                            "modifier": [],
                            "customer":
"593664ad84ae0a0a3feec056"
                        }
                    ],
                    "type": "smarthome"
                }
            ]
        },
        "status": "ok"
    }
```


slots中的“control_obj”即要操作的设备，上面的结果可以看到需要操作的设备是”灯”，动作为”打开”。应用程序根据这两个信息就可以在自己的设备中寻找“灯”这个设备，并发出“打开”命令。

输出ZigBee 3.0 协议

根据NLI的输出我们可以判定要控制的设备是灯，而灯的cluster我们选择了`ZCL_CLUSTER_ID_GEN_ON_OFF`，根据这个cluster以及等的device ID等输出命令即可。

连接硬件

这里没有提供驱动硬件的代码，但基本流程就是，将ZigBee协调器的开发版通过串口和电脑相连，软件发出的命令经串口发送给协调器，再由协调器控制ZigBee协议即可。

APP功能

文字输入

通过设备选择可以切换不同的例句。同时，可以在例句的框里输入其他控制语句，按回车可以重复输入。比如：“请帮我打开灯”，“灯给我打开”，“开一下空调”，“空调的温度提高一点”

语音输入

点击”开始录音”，如果没有点击“停止录音”，3秒之后会自动停止录音。如果在这之前点击了“停止录音”，那么会及时停止录音，并进行语音识别。

识别后的文字会显示在按钮的上方，如下图所示：

设备模拟

如上图所示，应用程序中会模拟彩灯的颜色和空调的温度、模式、风力，其原理就是根据输出的Zigbee3.0协议进行显示。

命令输出

即输出ZigBee3.0的协议。下面列出例子中的几种设备的协议信息。

灯

功能：仅支持打开和关闭

Device Dype: 0x100

命令：

Cluster ID: 0x0300

Cluster ID 的TI定义：ZCL_CLUSTER_ID_GEN_ON_OFF

actionID	Action_frame(1 bit)	参数组	说明
0x00	0x01	无	Off,关闭
0x01	0x01	无	On,打开

彩灯

功能：打开，关闭，颜色调节（例子仅支持红、橙、黄、绿、青、蓝、紫），氛围调节，色调调节。比如运动氛围、浪漫氛围、冷色调、暖色调等。

Device Dype:0x0102

命令：

Cluster ID: 0x0006

Cluster ID 的TI定义：ZCL_CLUSTER_ID_GEN_ON_OFF

actionID	Action_frame(1 bit)	参数组	说明
0x00	0x01	无	Off,关闭
0x01	0x01	无	On,打开

Cluster ID: 0x0300

Cluster ID 的TI定义：ZCL_CLUSTER_ID_LIGHTING_COLOR_CONTROL

actionID	Action_frame(1 bit)	参数组	说明
0x08	0x01	Attr1,Attr2	(均为int16，即两个字节，数据格式编号为0x29) 设置彩灯的颜色，即R,G,B值。

第一个参数的高八位表示R值。

第一个参数的低八位表示G值。

第二个参数的高八位表示B值。

第二个参数的低八位无意义。

电视

功能：打开，关闭，提高降低音量，换台。

Device Dype:0x0006

命令：

Cluster ID: 0x0006

Cluster ID 的TI定义：ZCL_CLUSTER_ID_GEN_ON_OFF

actionID	Action_frame(1 bit)	参数组	说明
0x00	0x01	无	Off,关闭
0x01	0x01	无	On,打开
0x05	0x01	无	提高音量
0x06	0x01	无	降低音量

Cluster ID: 0x0008

Cluster ID 的TI定义：ZCL_CLUSTER_ID_GEN_LEVEL_CONTROL

actionID	Action_frame(1bit)	参数组	说明
0x00	0x01	1个字节，uint8	切换频道
0x01	0x01	无	切换下个频道
0x02	0x01	无	切换上个频道

空调

功能：

开关功能，即打开和关闭空调。

切换模式，顺序为“自动-制冷-除湿-送风-加热”模式按顺序循环切换，但不支持某个模式的设置。

风力切换，切换顺序为“自动-低速-中低速-中速-中高速-高速-超强”。

其中，制冷和制热模式支持上述7种风力切换。

送风和自动模式没有“超强”风力

除湿无风力调节。

注意，仅支持切换，无风力设置。

升高温度，切换一次，温度上升一度，基础范围是16-30.

降低温度，每切换一次，温度下降一度，基础范围是16-30

状态查询，开关、温度、风力、模式查询。

Device Dype:0x0102

命令：

Cluster ID: 0x0300

Cluster ID 的TI定义：ZCL_CLUSTER_ID_LIGHTING_COLOR_CONTROL

actionID	Action_frame(1bit)	参数组	说明
0x00	0x01	无	切换开关
0x01	0x01	无	切换模式
0x02	0x01	无	切换风速
0x03	0x01	无	Setup Button
0x04	0x01	无	Setdown Button

窗帘

功能：

打开，关闭，停止运行，指定窗帘运行的位置。

Device Dype: 0x202

命令：

Cluster ID: 0x102

Cluster ID 的TI定义：ZCL_CLUSTER_ID_CLOSURES_WINDOW_COVERING

actionID	Action_frame(1bit)	参数组	说明
0x00	0x01	无	打开
0x01	0x01	无	关闭
0x02	0x01	无	窗帘电机停止移动，并返回当前位置的百分比

actionID	Action_frame(1bit)	参数组	说明
0x04	0x01	窗帘号房间号,2bytes	设置窗帘号，房间号.其中高8位是房间号，低8位是窗帘号
0x05	0x01	百分比	单位百分比

其余设备和传感器的ZigBee 输出协议不再一一列出，可以直接在APP中测试。

APP测试

代码下载解压之后，可以在根目录找到 smarthome.jar,在windows7 环境下双击即可以运行。

应用程序支持的语料除了选项里的，其他的相似说法也支持。

APP源码解析

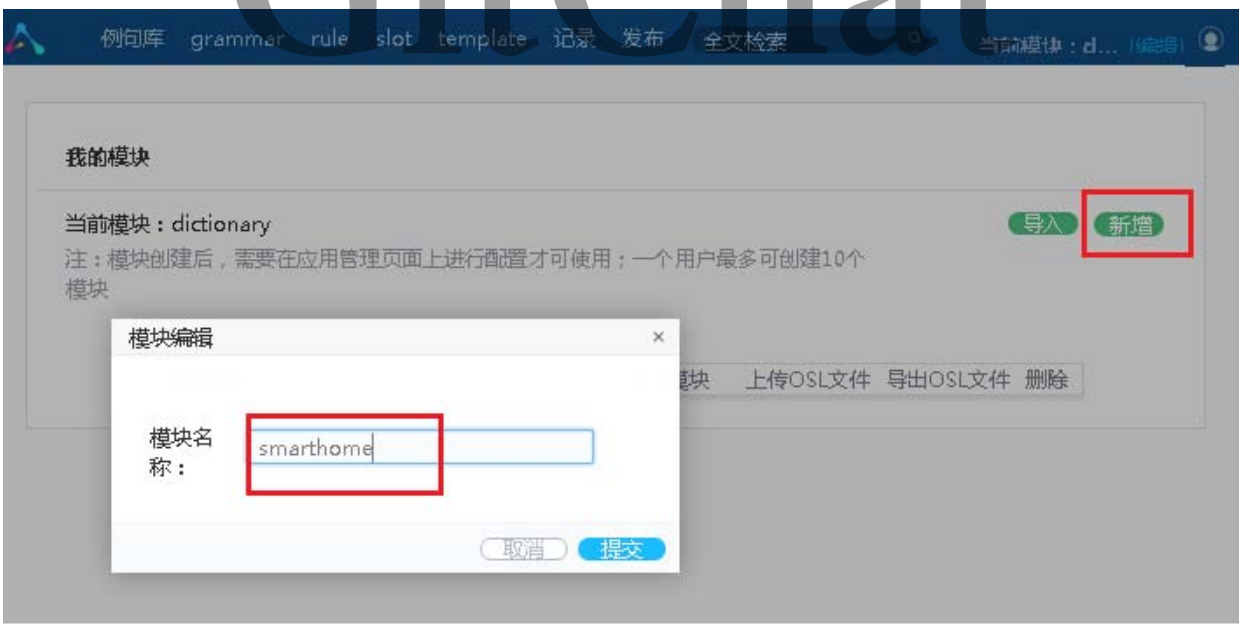
OLAMI语法加载

因为APP调用了OLAMI的自然语言理解接口，所以首先是必须先写语法，来匹配智能家居控制语句。比如：“打开灯”，“帮我打开空调”，必须在完成语法之后，才能从OLAMI的接口中获取NLI结果。语法相关定义和写法等请参考博客：告诉你如何使用OLAMI自然语言理解开放平台API制作自己的智能对话助手

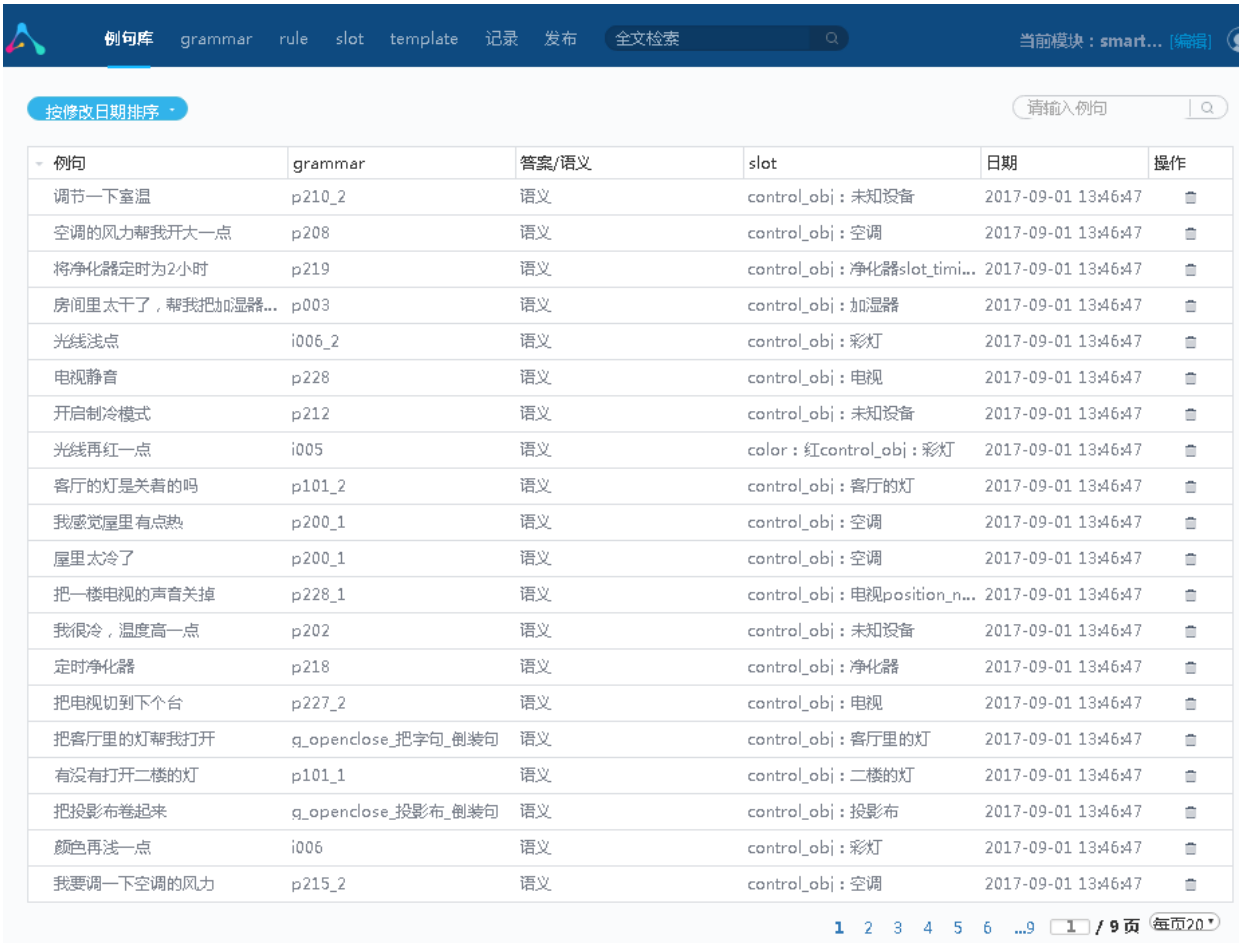
如果你希望修改语法，添加更多的句子支持，必须将语法文件导入到欧拉蜜NLI系统。

下载包解压之后，根目录找到**smarthome.osl**，这个就是智能家居支持的语法。然后注册并登录欧拉蜜官网,在自己的账号下找到“应用管理”，并进入NLI系统。如下图所示。

接着新增模块，并将智能家居语法smarthome.osl导入，如下图所示，点击“新建”并输入APP的名字“smarthome”,这个名字必须与smarthome.osl的名字相同，否则导入时会报错。当然也可以修改，但同时要修改smarthome.ols中APP name相关字段。



模块创建之后，选择“上传OSL文件”，然后选择smarthome.osl并确认即可。上传成功之后会进入该模块内部，然后在例句库中可以看到很多智能家居控制的句子，同时也可以查看Grammar,Rule等。至此OLAMI语法加载完毕。

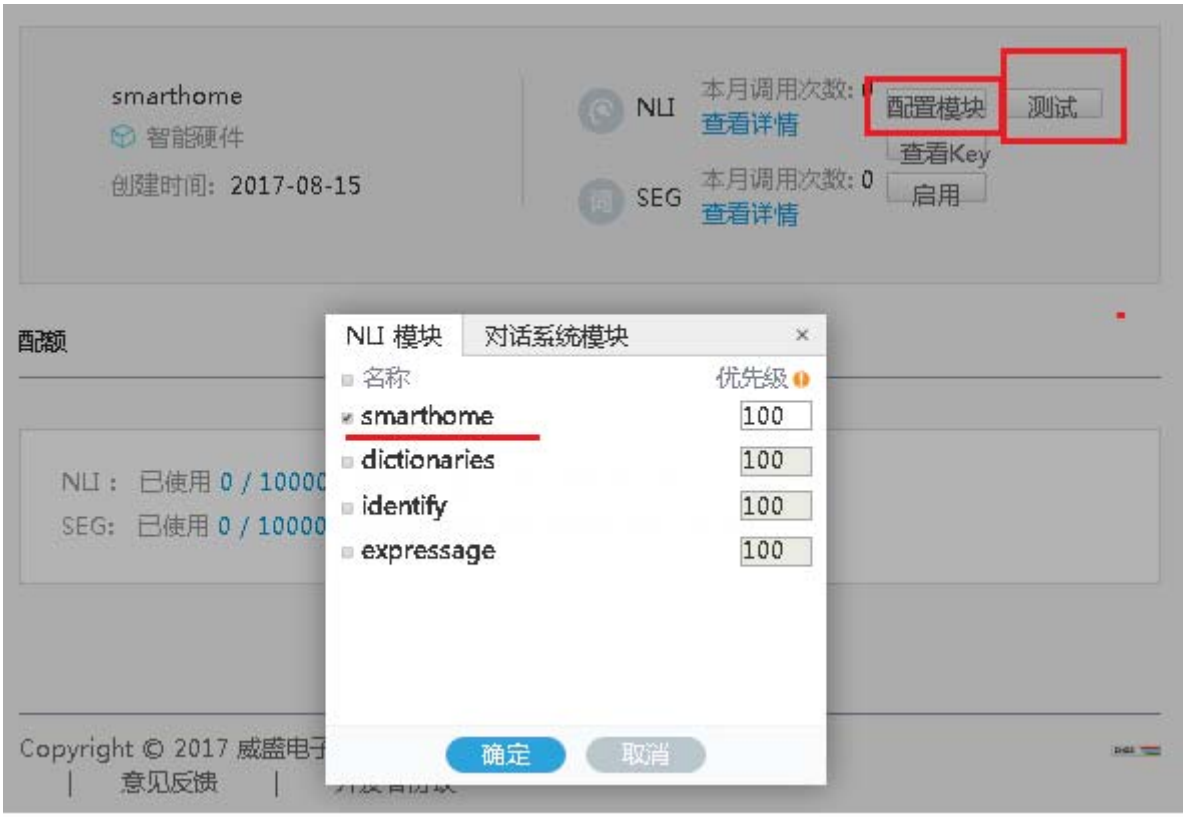


获取应用程序的APP KEY和APP Secreat

如果希望获取句子解析后的结果，必须在欧拉蜜平台中创建自己的应用程序，名字任意，我的叫“smarthome”。

回到“应用管理”界面——创建应用程序。

应用程序创建成功之后，还需要把刚才创建的**smarthome 语法模块**添加到应用程序中，一个应用程序可以支持多个语法模块。



点击图中的“测试”，输入“打开灯”，就可以看到JSON格式的语义输出结果了：

语法模块配置好之后，点击应用程序的”查看Key”的按钮，可以看到平台分配的APP Key和APP Secret.

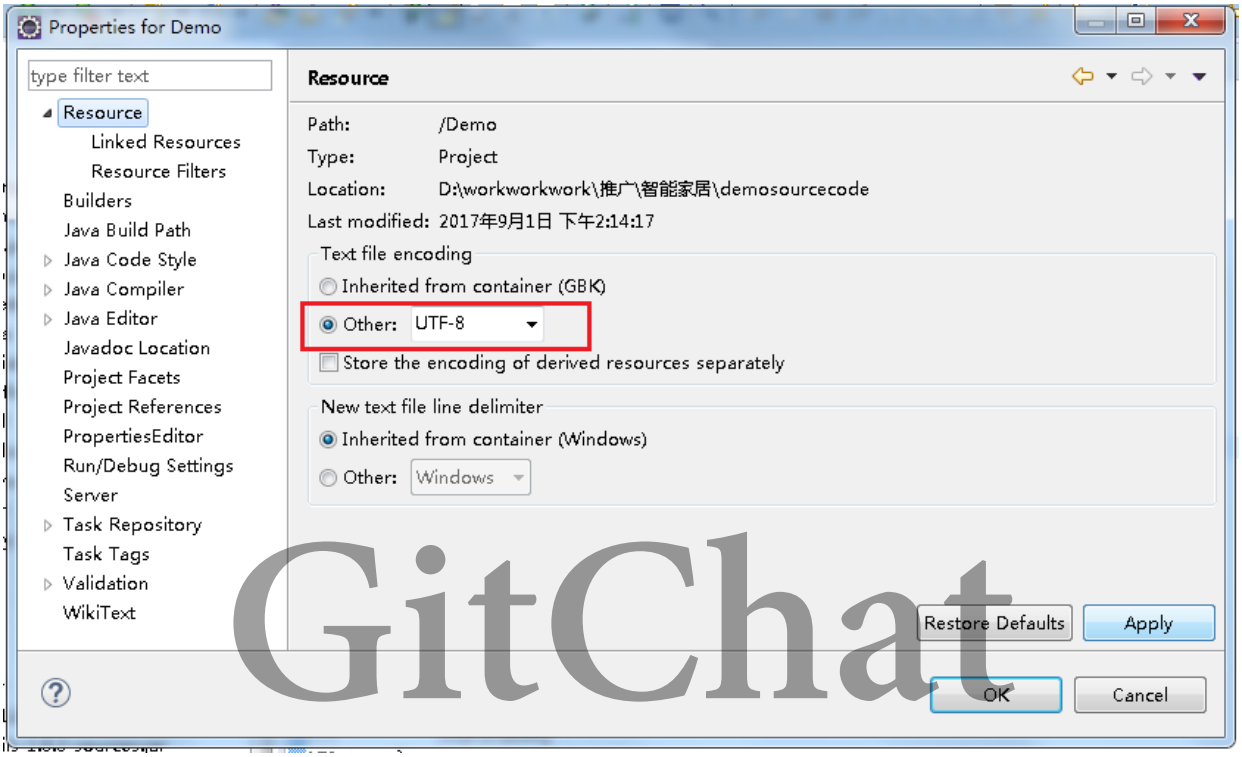
源码分析

源码工程是demosourcecode.jar，解压之后，添加入Eclispe工程，我的开发环境是JDK1.8.

Eclipse Version: Mars.2 Release (4.5.2).

注意：导入工程后，如果出现文字报错，请将默认编码修改为UTF-8,方法 Project->Properties->Resource

代码结构：



替换KEY

在smarthome packge中的NLIProcess.java中，替换之前创建语法应用时的APP Key和APP Secreat:

```
    // * Replace your APP KEY with this variable.
          private static String appKey = "*****your APP
Key******";

    // * Replace your APP SECRET with this variable.
           private static String appSecret = "****your APP
Secret*****";
```

程序入口：

程序入口为smarthome packge下的window.java,可以安装windows Builder插件，直接操作界面。

smarthome packge：

smarthome包里的源码包括了APP应用的基本框架，其中：

window.java为APP入口，即界面。

NLIProces.java表示处理来自OLAMI NLI接口的语义结果.

录音处理为：getSemanticBySpeech()

文字处理为：getSemanticByText(String inputText)

windowVariable.java是window.java和NLIProces.java的数据传递媒介， window.java中会将
NLIProcess.java 需要的控件传过去:

```
private void initialize() {
        nliwindowdata.setCmdTable(cmd_table);
        nliwindowdata.setcolortext(color_text);
        nliwindowdata.setAnswerText(answer_Text);
        nliwindowdata.setModetext(mode_text);
        nliwindowdata.setTempetext(tempe_text);
        nliwindowdata.setVoicetext(voice_text);
        nliwindowdata.setWindtext(wind_text);
        nliwindowdata.setisRed(isred);
        nliprocess.SetAnswerConfigCom(nliwindowdata);
```


smartHomeApp.java用来处理智能家居APP的语法解析和命令输出。是NLIProces.java中其中一个小模块。 你还可以在NLI处理中添加其他处理模块，比如天气查询、诗歌背诵等等。目前NLIprocess.java中仅处理了smarthome相关的NLI输出：

```
private void ProcessNLIResults(NLIResult[] nliResults) {
    // TODO Auto-generated method stub
    String  answer="对不起，你说的话我还不能理解";
    boolean isnormal=false;
    for(int i=0;i<nliResults.length;i++){
        NLIResult tempNlI=nliResults[i];
        //tempNlI.
        //voice_text

if(tempNlI.getType()!=null&&tempNlI.getType().equals(nliDefinitio
ns.smarthome_app)){
```

APPSlotEntry.java—-处理NLI返回的JSON数据中slots相关信息

OutputMap.java——存放smartHomeApp.java返回给NLIProcess.java的输出语句和命令。

Smarthome.definition packge

该包是智能家居处理中用到的定义和设备状态解析。

Demo中模拟了灯，彩灯，电视，空调，传感器等设备，初始化数据见smartHomeApp.java的InitDeviceData()。

所有设备信息通过ClientHomeAutomation.java解析并存储。

```
        //key is deviceID
            Map<String,HomeAutodeviceObjectNew> addedDeviceMapNew=new
    ConcurrentHashMap<String,HomeAutodeviceObjectNew>();
```

Smarthome.util

DataBuffer.java 和Microphone.java用来进行麦克风录音；录音格式按照欧拉蜜平台的要求,参数为16位深采样率，16KHZ频率，单声道。

源码为

```
        public Microphone() {

                this.sampleRate = 16000;
                this.bigEndian = false;
                this.signed = true;

                this.desiredFormat = new AudioFormat
                        (sampleRate, 16, 1, signed, bigEndian);
                //this.closeBetweenUtterances =
    closeBetweenUtterances;
                this.msecPerRead = 100;
                //this.keepDataReference = keepLastAudio;
                //this.stereoToMono = stereoToMono;
                //this.selectedChannel = selectedChannel;
                //this.selectedMixerIndex = selectedMixerIndex;
                this.audioBufferSize = 9600;

                recorderData = new DataBuffer();
            }
```


麦克风的录音开始和停止通过线程监控完成。直到没有声音录入时，录音线程才会触发录音停止机制，因此希望停止录音时必须通Microphone.stopRecording()关闭录音，程序才能停止录音。

因此录音时最好设置默认的录音时长或者通过标志来停止录音，并调用Microphone.stopRecording()，我这里的默认录音时长为3s.

代码见NLIProcess.java的

```
//最多录3秒数据，因为采样频率是16000点每秒，每个点占两个字节。
    // readcount<=0表示录音结束
    int num=0;
    int srcint=0;
    while(readcount > 0 )
    {
        if(total_count >= 48000*2||needstop)
            break;

        num++;
        System.out.println("数据"+(num+1));
        total_count += readcount;
        System.out.println("单数"+readcount);

        speechrecoginzer.appendAudioFramesData(databytes);

        readcount = mic.getData(databytes, 0, temsize);
    }

    mic.stopRecording();
```

WaveFileWriter.java可以为录音数据添加wav头。

和硬件设备对接

和硬件设备对接，需要串口或者USB等将输出的ZigBee协议发给协调器，由协调器控制各智能设备做出反应。