青 岛 科 技 大 学

综合课程设计报告

学生班级 机械 151

学生学号 1505160123

学生姓名 赵靖石

2018 年 6 月 29 日

目录

1 需求分析

1.1 系统简要分析

人工智能对话系统（以下简称本系统）是一个运行在 linux 系统上的程序，支持 X86，ARM 等大多数主流 CPU 架构。主要功能是实现基础的语音对话交流，并可以通过语音执行一些自定义命令。本系统仅实现底层和一些简单的自定义命令例子。UI 界面仅为命令行。

1.2 基本需求功能点分析

本系统，主要目标是简化人机交互过程，使用简洁的语音命令代替复杂的图形化界面，提高人机交互效率。降低用户的使用门槛，尽量减少用户的交互流程和难度，让产品的使用过程接近用户的自然行为。用户通过说话即可完成唤醒、查询、关闭和一系列复杂的人机语音交互操作。

1） 系统用户主要分为两大类：

1. 管理员用户类（相当于一名有各种操作权限的超级用户，该用户可以在本系统的基础上添加新命令）

2. 普通用户类（本系统的受众用户，可以通过语音交互来让系统执行一些已经定义好的命令）

2） 系统性能：

新命令添加，数据查询高效准确，提高工作人员的管理效率。

3） 系统输入：

1. 管理员：鼠标键盘等 PC 输入设备。

2. 普通用户：语音输入，必要时可以添加图像输入。

1.3 系统运行条件分析

此软件系统需要至少一台计算机或运行 linux 系统的单片机，并要求该计算机连接到 Internet，详细系统要求如下：

Linux2.6 或更高版本（本系统测试使用的为 Ubuntu16.04 和 Debian）

Python2.7 或更高版本

Gcc4.8 或更高版本

Make 工具

Sqlite3 数据库

理论上同样可以运行在 window 上，但并未测试，以下为在 window 上运行所需要的工具

Window xp 或更高版本

Python2.7 或更高版本

Gcc4.8 或更高版本

Sqlite3 数据库

Make 工具

各种依赖包的动态链接库

我没想到但是还需要的东西

2 系统设计

2.1 系统功能模块设计

此部分主要描述各个模块具体实现的各项功能点以及模块之间的关系，模块结构划分如图 2-1 所示：

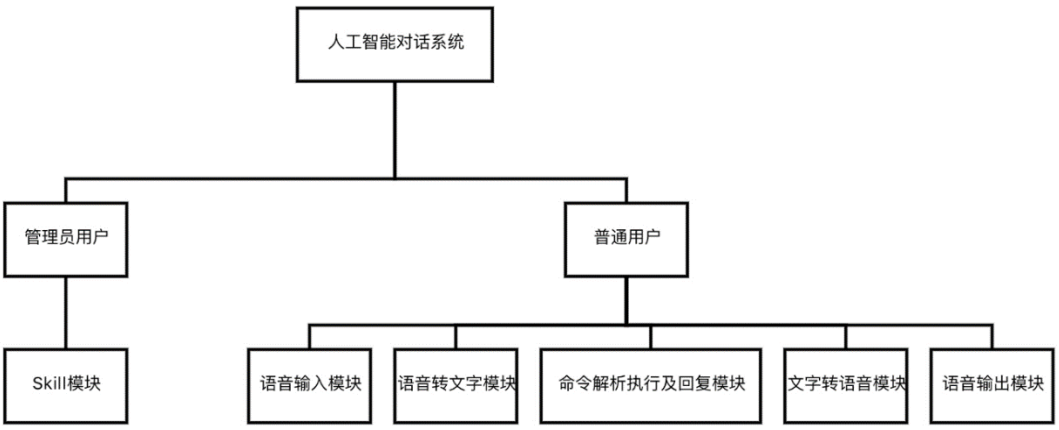

图 2-1 系统模块结构图

Fig.2-1 System module structure diagram

2.2 管理员用户模块

该模块具体功能是对本系统所掌握的技能进行管理和添加。

作为管理员，可以按照模板添加新的技能（Skill）供普通用户调用，或添加命令。

2.3 普通用户功能模块

普通用户的功能模块主要包括：语音输入模块，语音转文字模块，命令解析模块，命令执行及回复模块，文字转语音模块，语音输出模块等。

1. 语音转文字模块和文字转语音模块为调用百度等云平台服务，但应当规范为相同或类似的接口函数。

2. 语音输入模块和语音输出模块为对系统调用中语音设备调用的封装。

3. 命令解析模块主要解析用户命令并转化为标准的意图。

4. 命令执行及回复模块主要按照意图查找对应的技能，并执行该技能对应的函数，然后返回要展示给用户的命令。

2.4 系统功能设计

本部分按系统主要功能绘制流程图，说明系统功能将如何实现。

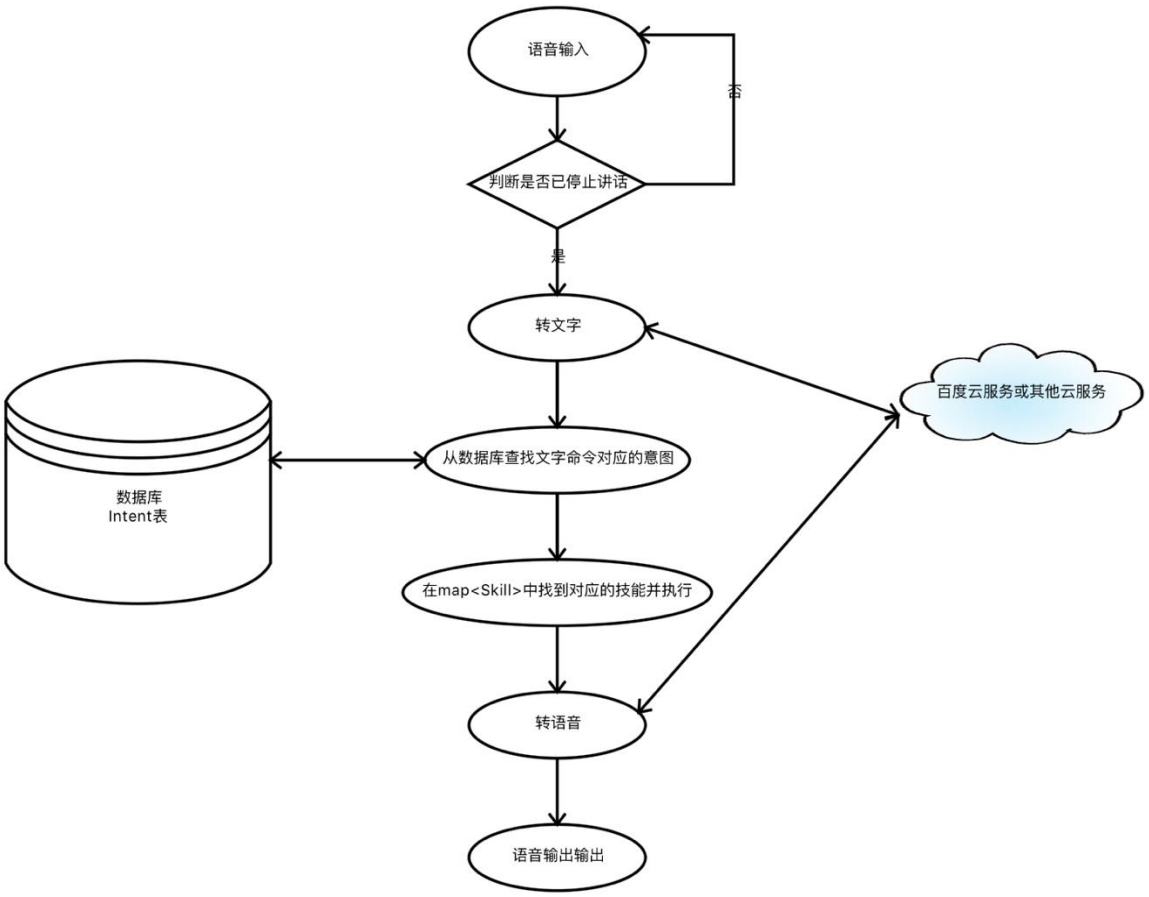

图 2-2 主要功能流程图

Fig.2-2 Main function flow diagram

2.5 系统依赖示意图

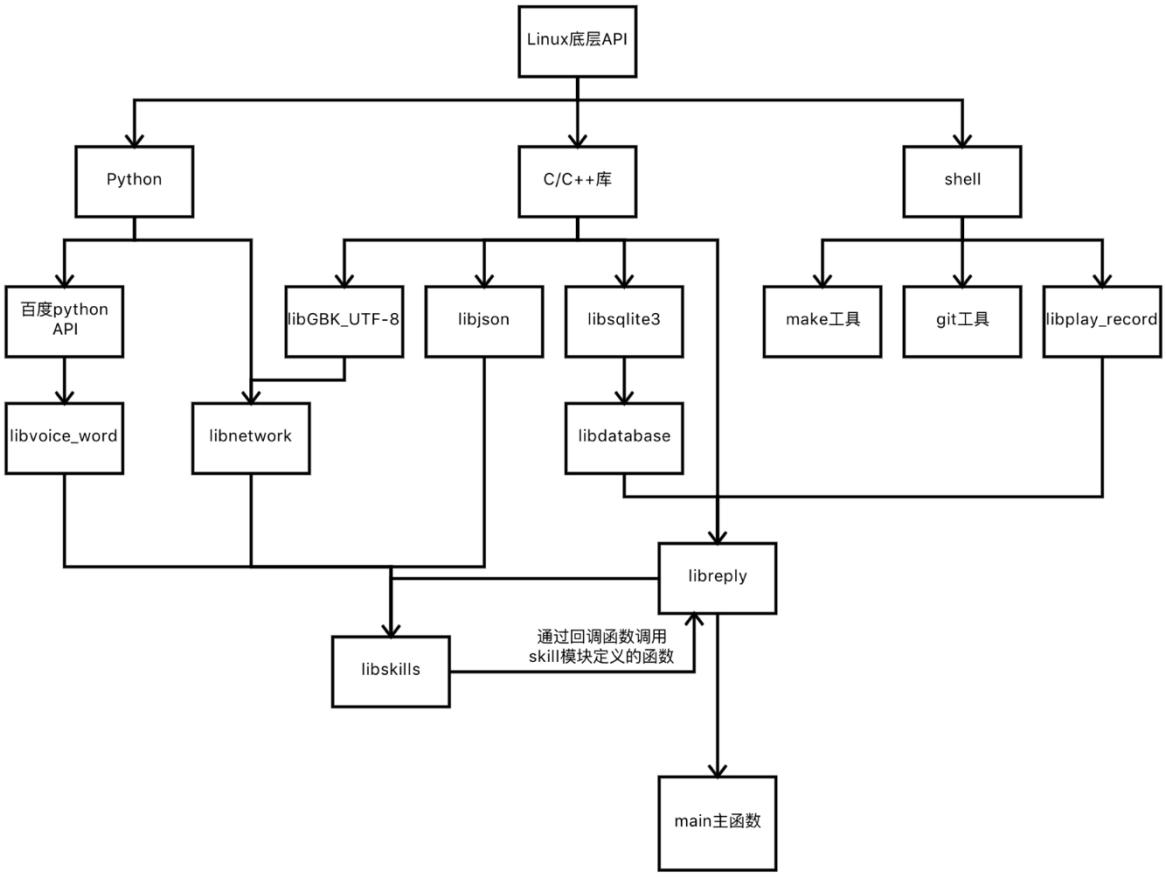

图 2-3 系统依赖示意图

Fig.2-3 System Dependency Diagram

3 数据库设计

3.1 概念结构设计

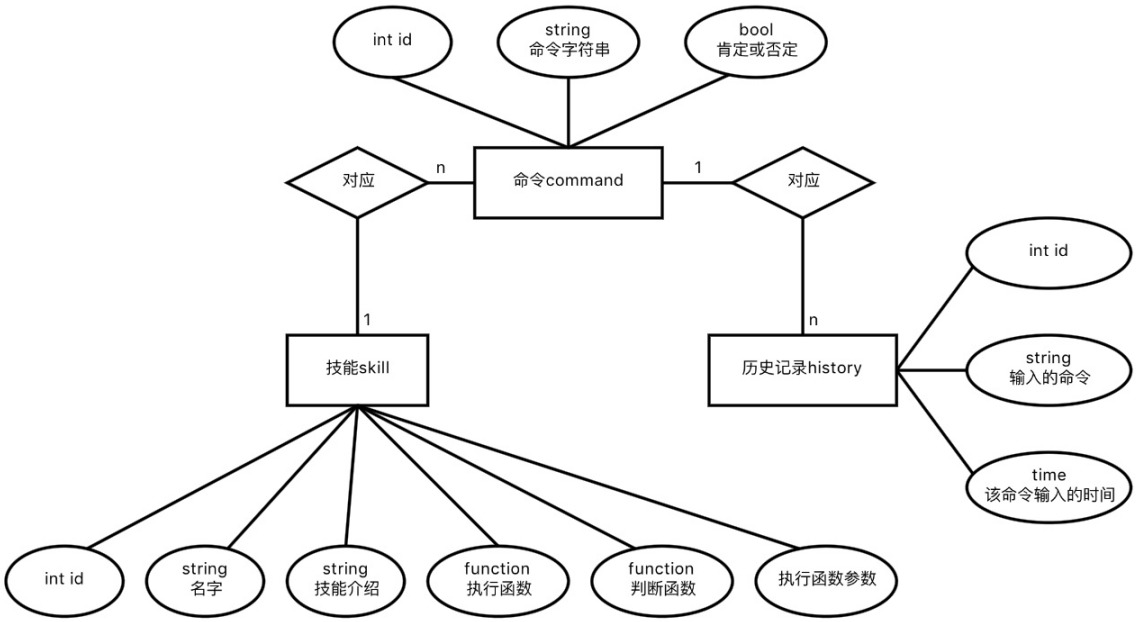

图 3-1 数据库设计 E-R 图

Fig.3-1 Database Design E-R Diagram

3.2 逻辑结构设计

3.2.1 命令表，具体数据如表 3-1 所示：

表 3-1 命令表

Tab.3-1 Commands table

列名	数据类型	长度	约束	说明
id	int	4	主码	命令 id
cmdString	varchar	50	非空	命令字符串
isPositive	bool	1	非空	命令的肯定性

3.2.2 关系表，具体数据如表 3-2 所示：

表 3-2 关系表

Tab. 3-2 relations table

列名	数据类型	长度	约束	说明
id	int	4	主码	关系 id
skill	int	4	非空	对应的技能 id
command	int	4	非空	对应的命令 id

3.2.3 历史记录表，具体数据如表 3-3 所示：

表 3-3 历史记录表

Tab. 3-3 history table

列名	数据类型	长度	约束	说明
id	int	4	主码	历史记录 id
time	varchar	50	非空	本命令的输入时间
command	varchar	50	非空	输入的命令

3.2.4 技能结构体和 skills 图，存储在内存里，具体结构如下所示：

对于技能结构体，map 中对应的 int 为 id，相当于 skill 的主键，execFunc 为当输入为符合条件的字符串时执行的函数，judgeFunc 为判断输入字符串是否符合条件的函数，introduce 存放的为技能介绍，numP 为执行函数参数数量，pList 为执行函数参数列表。

```
extern int _skillId;//最后一个加载的技能的 id 号，同时代表技能的优先级，号越小优先级越高，需要在 init 时初始化为 0

extern const int MaxSkillId;//技能未定义指令的 skillId,指示了技能的最大数量


typedef struct skill_t{
    int id;//自动生成
    bool (*judgeFunc)(std::string cmd);//判定函数,判断是否符合执行条件,当然,您也可以在这里面顺便填充 pList
    std::string (*execFunc)(int numP,std::list<long> pList);//执行函数,返回值是返回的回复
    int numP;
    std::list<long> pList;//执行函数的输入参数列表,您只能输入整数,并且应当处理所有情况的输入
    std::string name;
```

```
    std::string introduce;
} skill_t;

extern std::map<int,skill_t> skills;
```

3.2.5 外部设备结构体(暂未实现)

```
extern int _deviceId;
extern int maxDeviceId;

typedef struct device_t{
    int id;
    std::string ip;
    std::string name;
    std::string introduce;
} device_t;

extern std::map<int,device_t> devices;
```

3.3 数据库建表脚本

```
CREATE TABLE 'command' (
    `id`    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    `cmdString` TEXT NOT NULL UNIQUE,
    `isPositive`    INTEGER DEFAULT 0
);
CREATE TABLE `relation` (
    `id`    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    `intent`    TEXT NOT NULL,
    `command`   INTEGER NOT NULL
);
CREATE TABLE 'histroy'  (
```

```
    `id`    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    `command`    TEXT NOT NULL,
    `time`    TEXT NOT NULL
);
```

3.4 数据库的选择

起初计划用 mysql 数据库，后来转向了 sqlite

原因：

SQLite 是一个轻量级、跨平台的关系型数据库。

它有以下几个重要特点：

1） 轻量级

SQLite 和 C/S 模式的数据库软件不同，它是进程内的数据库引擎，因此不存在数据库的客户端和服务器。使用 SQLite 一般只需要带上它的一个动态 库，就可以享受它的全部功能。而且那个动态库的尺寸也挺小，以版本 3.6.11 为例，Windows 下 487KB、Linux 下 347KB。

2） 绿色

它的核心引擎本身不依赖第三方的软件，使用它也不需要“安装”。所以在部署的时候能够省去不少麻烦。

3） 单一文件

所谓的“单一文件”，就是数据库中所有的信息（比如表、视图、触发器、等）都包含在一个文件内。这个文件可以 copy 到其它目录或其它机器上，也照用不误。

4） 跨平台/可移植性

支持主流操作系统。而且能在嵌入式系统中使用。

4 系统实现

4.1 系统项目清单

系统项目具体所含如图 4-1 所示：

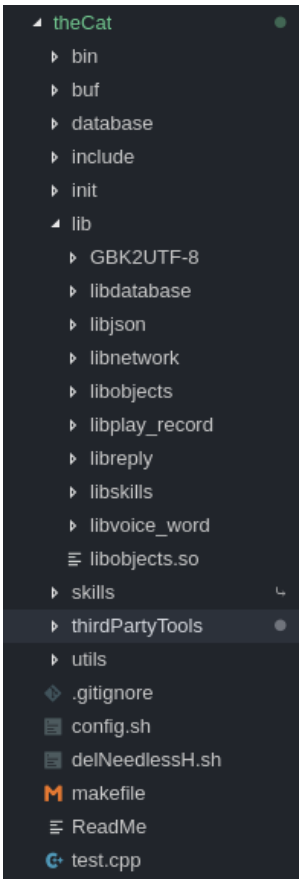

图 4-1 系统项目结构图

Fig. 4-1 System project structure

4.2 可执行文件目录 bin

该目录下存放 make 生成的最终可执行文件。

4.3 缓存文件目录 buf

该目录用于存放缓存文件。包括 python 脚本文件 py、语音识别生成的缓存文本 txt、语音合成生成的语音 mp3、网络请求返回的 json 或 xml 等。

原则上，所有缓存文件都应该放在该目录下。

4.4 数据库文件目录 database

该目录用于存放与数据库有关的文件。

4.4.1 build.sql 用于初始化数据库的 sql 脚本文件。该文件会在 database 模块构建时复制到该目录下。

4.4.2 database.db 项目的数据库文件。该文件会在 database 模块中的 DB_build 函数执行时生成。

4.5 头文件目录 include

该目录存放了该项目中所有的头文件或头文件副本。执行 make copy_h 时将各.h 文件从各子目录复制到该目录下。

4.6 初始化模块目录 init

该目录包含了第一次构建时应当执行的脚本，用于配置该项目和安装依赖包。

4.7 库文件目录 lib

该目录是整个项目的核心部分，存放了大多数使用到的模块的源代码，各模块生成的动态链接库.so 会存放在该目录下。

4.7.1 字符转换模块 GBK_UTF-8

该模块实现了从 GBK 到 UTF-8 编码的转换，和从 C 语言的\x 格式 UTF 向%格式 UTF 的转换。主要在网络通讯时使用。

该模块的主要接口函数如下：

1) std::string GBK2UTF_8(std::string gbk);

将 GBK 字符串转换为 UTF-8 字符串。

2) std::string UTF_82GBK(std::string utf8);

将 UTF-8 字符串转换为 GBK 字符串。

3) std::string UTF_82GBK_X(std::string utf8,bool upper=true);

将 UTF 字符串转换为以%分割的 GBK 字符串。

参数：upper 返回的字符串里十六进制数 a-f 是否为大写字母。

4) std::string GBK2UTF_8_X(std::string gbk,bool upper=true);

将 GBK 字符串转换为以%分割的 UTF 字符串。

5) int u2g(const char *inbuf,char *outbuf,size_t outlen);

int g2u(const char *inbuf,char *outbuf,size_t outlen);

未经规范的函数原型，不建议在外部直接调用，但在需要更多编码转换功能时可以使用。

4.7.2 数据库模块 libdatabase

该模块实现了对 sqlite 数据库的操作。

该模块的主要接口函数如下：

1） void DB_init(std::string filename = "$CAT_HOME/database/database.db");

设置将要操作的数据库文件的路径。如果不存在，将会创建。

2） void DB_build(std::string build_filename = "$CAT_HOME /database/build.sql");

执行 build.sql 来创建数据库中的表。

3） bool DB_exec(std::string cmd,callbackFunc _callback = NULL);

执行一条或多条 sql 命令，各命令间用;隔开。

参数：

cmd 要执行的命令

_callback 回调函数

4） void DB_displayTable(std::string tableName);

展示某个表格的内容。

输出格式：第一行为列名，之后为数据，各列间用制表符隔开，各行间用换行符隔开。

5） void DB_showRelations();

展示 relation 表的内容。

6） void DB_insertTestValues(std::string cmdString,std::string intent);

插入测试数据。

7） void DB_addHistroy(std::string cmd,std::string time);

添加一条历史记录到 history 表。

8） void DB_showHistroys();

展示历史记录表的内容。

9） int display_callback(void* para,int columnCount,char** columnValue,char** columnName);

用于展示表的回调函数。

4.7.3 json 文件处理模块 libjson

该模块为三方库，实现了对 json 文件的处理。

4.7.4 网络模块 libnetwork

该模块实现了发送 get 请求到指定 ip 的函数。之后会实现发送 post 等其他请求。

该模块的主要接口函数如下：

void doGet(std::string host,std::string para="",std::string response="./res.txt",int port=80);

发送 Get 请求到 host 的 port 端口，请求的参数为 para，并将返回信息存放在 response 指定的文件中。

4.7.5 认知模块 libobject（暂未使用）

该模块存放与可以认知到的 object 以及认知到的对象之间的关系，或与认知相关的其他函数。

4.7.6 语音模块 libplay_record

该模块实现了语音播放和录音功能。

该模块的主要接口函数如下：

1) void play(std::string filepath="./buf/buf.pcm");

播放函数。播放指定文件。

2) void record(int time,std::string filepath="./buf/buf.pcm");

录音函数。录音 time 秒并将录音所得文件保存在指定位置。

4.7.7 回复模块 libreply

该模块实现了与回复有关的函数。该模块为整个项目核心中的核心。其中，核心函数包括 listen、say、reply。

该模块的主要接口函数如下：

1) bool inline rule_include(string inputStr,string str);

内联函数。定义了判断规则“包含”。

判断 inputStr 字符串是否包含 str 字符串。

2) bool inline rule_include_inOrder(string inputStr,string str0,string str1);

内联函数。定义了判断规则 “包含并且按顺序出现” 。

判断 inputStr 中是否同时包含字符串 str0 和 str1 并且 str0 出现在 str1 之前。

3) string ask(string question,int waitTime=5,string inputFile="./buf/buf.pcm",string outputFile="./buf/result.mp3");//询问并直接获得结果

核心函数之一。询问函数。

先问一个问题，然后获得用户得回答。

4) string listen(int waitTime=5,string inputFile="./buf/buf.pcm");

核心函数之一。倾听函数。

获得用户的语音输入并转换为字符串。

inputFile 为录音缓存文件的路径。

5) void say(string str,string outputFile="./buf/result.mp3");

核心函数之一。说函数。

将字符串转换为语音并播放。

outputFile 为转语音缓存文件的路径。

6) void replyInit();//add the basic skills to SKILL moudle

初始化 reply 模块。在启动时将基础的技能动态加载进系统。

7) string reply(string inputStr);

核心函数之一。回复函数。

将输入字符串按照各技能模块的判断函数进行判断，并执行第一个符合条件的技能的执行函数，然后返回执行函数的返回值。

4.7.8 语音文字转换模块 libvoice_word

该模块实现了语音转文字和文字转语音函数。本质上是对第三方语音转换 API 的规范化。目前只添加了百度语音识别，之后会考虑加入其他，如谷歌和科大讯飞。

该模块的主要接口函数如下：

1) std::string voice2word(std::string filepath="test.pcm");

语音转文字函数。将指定语音文件转换为文字。

2) void word2voice(std::string text,std::string outputfile="result.mp3");

文字转语音函数。将指定文本转换为语音并存放在指定位置。

4.7.9 技能模块 libskill

该模块实现了技能加载卸载函数（skill 子目录）和技能的定义（其他子目录）。您可以在这个目录下定义新的技能并创建对应的子目录、判断函数和执行函数。

该模块的主要接口函数如下：

1) void skillInit();

初始化技能模块。其实只有把_skillId 置零。

2) bool loadSkill(skill_t skill,int skillId=(++_skillId));

加载一个新的技能到 skills 图。如果不输入 skillId，skillId 将会递增 1 来保证唯一性。

如果您要手动指定一个 skillId，请一定要保证 skillId 不会重复，否则后一个 skill 会被前一个覆盖。

3) bool unloadSkill(std::string skillName);

通过技能名来卸载一个技能。

4) bool unloadSkill(int skillId);

通过技能 id 来卸载一个技能。

5) bool unloadSkill(skill_t skill);

通过技能结构体来卸载一个技能。

注：实际上该函数只会关注 skill 的 key，即 skillId。

6) void DB_showSkills();

展示已经加载的技能。

4.7.10 设置模块 libsetting（暂未实现）

该模块将实现对全局设置 setting 修改的函数。

4.8 技能模块目录 skills

到 libskill 的链接。

4.9 三方库目录 thirdPartyTools

该目录下保存了用到的第三方库。

4.10 工具目录 utils

保存了用到的工具模块。分别链接到对应的 lib 目录。如 DButils 链接到了 libdatabase。

4.11 git 版本控制文件.gitignore

版本控制工具 git 生成的版本控制文件。

4.12 配置文件 config.sh（暂未使用）

用于配置本程序的脚本文件。主要用于配置环境变量 CAT_HOME 为项目根目录。

4.13 makefile

Make 工具的配置文件。

4.14 delNeedlessH.sh

用于删除多余头文件的脚本文件。在 makefile 中 make copy_h 命令中被调用。

4.15 说明文档 ReadMe

说明文档。后期会考虑改成 ReadMe.md。

4.16 主文件 test.cpp

内有主函数。整个项目的主函数入口。

内部逻辑：

1) 初始化各模块：建数据库，加载使用到的技能等。
2) 设置输入输出的缓存文件并显示。
3) 进入循环，要求输入您要录音的时间 waitTime，默认为 5 秒。
4) Listen waitTime 秒，并将识别出的文字传递给 reply，再将 reply 返回的回复传给给 say 以转成语音并播放。同时显示回复的字符串。
5) 获得当前时间，并将历史记录添加到数据库。
6) 返回第三步，循环。

5 总结

概述系统开发使用的技术，和系统完成的功能。

5.1 系统使用到的技术和完成的功能

5.1.1 选择的数据库为 sqlite3

5.1.2 用 make 工具实现了工程的自动化构建，以下为定义的 make 命令

通用部分：

1. make 构建

2. make clear 清除所有中间文件

3. make clean 清除所有生成文件(包括中间文件.o,动态链接库.so,可执行文件.out)

4. make run 运行构建完成的可执行文件,注:对生成文件为动态链接库的模块不适用

5. make test 构建并运行测试程序,运行后自动删除

根目录：

1. make 构建整体项目

2. make clear 清除所有模块的中间文件

3. make clean 清除整个项目中 make 生成的文件

4. make run 运行

5. make test 依次进入各个模块并执行所有的测试

6. make copy_h 将各个模块的头文件.h 复制到 include 文件夹,搜寻深度 2~3

5.1.3 实现了 skill 的动态加载和卸载。因为 map 的实现本质上是一个红黑树，对于 skill 的查找相当快。

5.1.4 核心函数的实现方法

1. listen 函数

```
string listen(int waitTime,string inputFile){
system(("touch "+inputFile).c_str());//创建缓存文件


cout<<"录音"<<waitTime<<"秒"<<endl;
record(waitTime,inputFile);//录音
cout<<"正在转语音..."<<endl;
string input=voice2word(inputFile);//转语音
cout<<"您的输入是:"<<input<<endl;
```

```
    return input;
}
```

2. say 函数

```
void say(string str,string outputFile){
    system(("touch "+outputFile).c_str());//创建缓存文件

    word2voice(str,outputFile);//文字转语音
    cout<<"正在将回复转换成语音..."<<endl;
    play(outputFile);//播放
    cout<<"本轮回复结束"<<endl;
}
```

3. reply 函数

```
string reply(string inputStr){
    string r="";//最终返回值

    map<int,skill_t>::iterator iter;//迭代器,其元素 first 为 map 的 key,second 为 map 的 value
    for(iter=skills.begin();iter!=skills.end();iter++){/遍历 skills map
        if(iter->second.judgeFunc==NULL)//未定义判断函数时直接跳过该 skill
            continue;
        if(iter->second.judgeFunc(inputStr)){//判断是否符合判断函数定义的条件
            if(iter->second.execFunc==NULL){//执行函数为空时返回执行失败
                r="执行失败,技能"+iter->second.name+"没有执行函数";
                goto reply_end;
            } else ;
            r=iter->second.execFunc(iter->second.numP,iter->second.pList);//符合条件且定义了执行函数时执行执行函数,并把参数数目和参数列表传递给执行函数
            goto reply_end;
        }
    }
reply_end://reply 函数结尾
```

```
return r;

}
```

5.2 系统的不足以及待实现功能

最大的不足是没有一个优美的 UI 界面。尽管不太需要 UI 界面。

还没有加入机器学习，之后有时间的话会在 skill 模块中加入学习的技能。

设置模块需要写，同时应该增加修改设置用的 API。

应该增加在局域网内控制其他设备的技能和一个全局的外部设备 map

5.3 系统的二次开发

系统的二次开发应主要集中在对新的技能的设计。

对自定义技能的设计应当注意以下几点：

您对 Skill 的定义必须符合条件，否则可能无法正常加载该技能。

您可以使用在解析规则中定义的函数来辅助您写判断是否符合执行条件的函数。

如果您要自定义 Skill 的 id 而不使用 loadSkill()自动生成的 id,请您务必保证该 id 号未被使用且永远不会被使用。而且必须小于 MaxSkillId 值。id 大于 MaxSkillId 的 Skill 将永远无法被调用到。

您对 judgeFunc 和 execFunc 的定义必须是正确的类型，并且应当尽量是 skillName_judgeFunc 的形式，而且，我强烈建议您对函数先声明再定义。

可以通过网络通讯模块向已定义的外部设备发送消息来实现远程控制的功能。

以下是两个例子：

5.3.1 你好

您可以这样定义判断函数和执行函数

```
bool hello_judgeFunc(std::string cmd){
 return rule_include(cmd,"你好");
}
std::string hello_execFunc(int numP,list<long> pList){
 return "你好，"+masterName+"\n";
}
```

当您需要加载该技能时，可以这样加载。

```
skill_t tmp;
 tmp.name="hello";
 tmp.judgeFunc=hello_judgeFunc;
 tmp.execFunc=hello_execFunc;
```

```
tmp.introduce="问好";
tmp.numP=0;
tmp.pList=list<long>();
loadSkill(tmp);
```

5.3.2 天气

您可以这样定义判断函数和执行函数

```
string readFileJson();//用于读取 json 文件的函数
bool weather_judgeFunc(std::string cmd){//当输入包含天气时执行
    return rule_include(cmd,"天气");
}
std::string weather_execFunc(int numP,list<long> pList){
    string answer="";
    answer=ask("您想要查询哪里的天气?",3);//询问并获得回答，等待时间 3 秒

    answer.resize(answer.size()-4);//4 是百度魔数，因为百度语音识别会多返回一个’，’和’\n’，用于去掉这两个多余的 UTF 字符
    string city=GBK2UTF_8_X(UTF_82GBK(answer));//转换为%格式
    cout<<"正在查询"<<answer<<"的天气..."<<endl;

    doGet("v.juhe.cn/weather/index",
    "cityname="+city+"&dtype=&format=&key=1218059ee4c3713769eed2335e8a77c6",
    "./buf/weather.json",80);//发送 Get 请求到对应的网址来查询天气，并将返回结果存入缓存文件
    return readFileJson();
}
```

您可以这样加载

```
    skill_t tmp;
    tmp.name="weather";
    tmp.judgeFunc=weather_judgeFunc;
```

```
tmp.execFunc=weather_execFunc;
tmp.introduce="查天气";
tmp.numP=0;
tmp.pList=list<long>();
loadSkill(tmp);
```

6 附录

6.1 附录 1:所有文件列表

.

```
├─── bin
├─── buf
│   └─── python
├─── config.sh
├─── database
│   └─── test4database
│       ├─── a.out
│       ├─── insertData.sql
│       ├─── sqlite3.h
│       └─── test.c
├─── delNeedlessH.sh
├─── include
│   ├─── database.h
│   ├─── GBK_UTF8.h
│   ├─── init.h
│   ├─── IntelligentObject.h
│   ├─── json
│   │   ├─── autolink.h
│   │   ├─── config.h
│   │   ├─── features.h
│   │   ├─── forwards.h
│   │   ├─── json.h
│   │   ├─── reader.h
│   │   ├─── value.h
│   │   └─── writer.h
│   ├─── json_batchallocator.h
│   ├─── network.h
```

```
|   ├── object.h
|   ├── person.h
|   ├── play.h
|   ├── record.h
|   ├── reply.h
|   ├── skill.h
|   ├── sqlite3.h
|   ├── voice2word.h
|   ├── weather.h
|   └── word2voice.h
├── init
|   ├── init.sh
|   └── libinit
|       ├── include
|       |   ├── init.h
|       |   ├── IntelligentObject.h
|       |   ├── object.h
|       |   └── person.h
|       ├── init.cpp
|       └── makefile
├── lib
|   ├── GBK2UTF-8
|   |   ├── GBK_UTF8.cpp
|   |   ├── GBK_UTF8.h
|   |   ├── makefile
|   |   └── test.c++
|   ├── libdatabase
|   |   ├── build.sql
|   |   ├── database.cpp
|   |   ├── database.h
```

```
|   |   ├── database_local.cpp
|   |   ├── makefile
|   |   ├── test4DB.c++
|   |   └── test.sh
|   ├── libjson
|   |   ├── include
|   |   |   └── json
|   |   |       ├── autolink.h
|   |   |       ├── config.h
|   |   |       ├── features.h
|   |   |       ├── forwards.h
|   |   |       ├── json.h
|   |   |       ├── reader.h
|   |   |       ├── value.h
|   |   |       └── writer.h
|   |   ├── json_batchallocator.h
|   |   ├── json_internalarray.inl
|   |   ├── json_internalmap.inl
|   |   ├── json_reader.cpp
|   |   ├── json_value.cpp
|   |   ├── json_valueiterator.inl
|   |   ├── json_writer.cpp
|   |   ├── makefile
|   |   └── sconscript
|   ├── libnetwork
|   |   ├── doGet.py
|   |   ├── makefile
|   |   ├── network.cpp
|   |   ├── network.h
|   |   ├── result.txt
```

```
|   |   ├── test4network.c++
|   |   └── test.sh
|   ├── libobjects
|   |   ├── bin
|   |   ├── include
|   |   |   ├── IntelligentObject.h
|   |   |   ├── object.h
|   |   |   └── person.h
|   |   ├── IntelligentObject.cpp
|   |   ├── makefile
|   |   ├── makefile_so
|   |   ├── object.cpp
|   |   ├── person.cpp
|   |   ├── ReadMe
|   |   └── test.c++
|   ├── libplay_record
|   |   ├── makefile
|   |   ├── play
|   |   |   ├── play.cpp
|   |   |   ├── play.h
|   |   |   ├── test4play.c++
|   |   └── record
|   |       ├── record.cpp
|   |       └── record.h
|   ├── libreply
|   |   ├── makefile
|   |   ├── reply.cpp
|   |   ├── reply.h
|   |   ├── test4reply.c++
|   |   └── test.sh
```

```
|   ├── libskills
|   |   ├── light
|   |   ├── makefile
|   |   ├── README
|   |   ├── skill
|   |   |   ├── skill.cpp
|   |   |   └── skill.h
|   |   ├── test4skill.c++
|   |   ├── test.sh
|   |   └── weather
|   |       ├── weather.cpp
|   |       └── weather.h
|   └── libvoice_word
|       ├── buf
|       ├── makefile
|       ├── python
|       |   ├── voice2word.py
|       |   └── word2voice.py
|       ├── voice2word
|       |   ├── test.c++
|       |   ├── test.sh
|       |   ├── voice2word.cpp
|       |   └── voice2word.h
|       └── word2voice
|           ├── test.c++
|           ├── test.sh
|           ├── word2voice.cpp
|           └── word2voice.h
├── makefile
├── ReadMe
```

```
├─── skills -> /home/dts/desktop/theCat/lib/libskills
├─── test.cpp
├─── thirdPartyTools
│    ├─── aip-python-sdk-2.0.0.zip
│    ├─── asr-linux-cpp-demo.zip
│    ├─── audio-recorder_2.0.2.orig.tar.xz
│    └─── jsoncpp-src-0.5.0.zip
├─── tree.txt
└─── utils
    ├─── DButils -> /home/dts/desktop/theCat/lib/libdatabase
    ├─── GBK2UTF-8_utils -> /home/dts/desktop/theCat/lib/GBK2UTF-8
    ├─── netUtils -> /home/dts/desktop/theCat/lib/libnetwork
    ├─── play_record -> /home/dts/desktop/theCat/lib/libplay_record
    └─── voice_word -> /home/dts/desktop/theCat/lib/libvoice_word
```

6.2 附录 2:重要接口函数列表及其声明位置

```
//database.h
void DB_init(std::string filename = "/home/dts/desktop/theCat/database/database.db");
void DB_build(std::string build_filename = "/home/dts/desktop/theCat/database/build.sql");
bool DB_exec(std::string cmd,callbackFunc _callback = NULL);

void DB_displayTable(std::string tableName);
void DB_showRelations();
void DB_insertTestValues(std::string cmdString,std::string intent);

void DB_addHistroy(std::string cmd,std::string time);
void DB_showHistroys();

int display_callback(void* para,int columnCount,char** columnValue,char** columnName);

//GBK_UTF8.h
```

```
std::string GBK2UTF_8(std::string gbk);
std::string UTF_82GBK(std::string utf8);
std::string UTF_82GBK_X(std::string utf8,bool upper=true);
std::string GBK2UTF_8_X(std::string gbk,bool upper=true);

int u2g(const char *inbuf,char *outbuf,size_t outlen);
int g2u(const char *inbuf,char *outbuf,size_t outlen);

//network.h
void doGet(std::string host,std::string para="",std::string response="./res.txt",int port=80);

//play.h
void play(std::string filepath="./buf/buf.pcm");

 //record.h
void record(int time,std::string filepath="./buf/buf.pcm");

//reply.h
using namespace std;

bool inline rule_include(string inputStr,string str);
bool inline rule_include_inOrder(string inputStr,string str0,string str1);

string ask(string question,int waitTime=5,
        string inputFile="./buf/buf.pcm",string outputFile="./buf/result.mp3");//询问并直接获得结果
string listen(int waitTime=5,string inputFile="./buf/buf.pcm");
void say(string str,string outputFile="./buf/result.mp3");

void replyInit();//add the basic skills to SKILL moudle
string reply(string inputStr);
```

```
//skill.h
void skillInit();

bool loadSkill(skill_t skill,int skillId=(++_skillId));
bool unloadSkill(std::string skillName);
bool unloadSkill(int skillId);
bool unloadSkill(skill_t skill);

void DB_showSkills();

//voice2word.h
std::string voice2word(std::string filepath="test.pcm");

//word2voice.h
void word2voice(std::string text,std::string outputfile="result.mp3");

//weather.h
bool weather_judgeFunc(std::string cmd);
std::string weather_execFunc(int numP=0,std::list<long> pList=std::list<long>());
```