

# tmdb

---



## 介绍

本项目为武汉大学2019级计算机弘毅班"数据库系统实现"课程的第三次大作业

组名: 啊对对对

- 组长:陆知行
- 组员:孙含笑
- 组员:陶文琪
- 组员:徐梓峻

Totem Mobile Database at WHU (This is a prototype system for teaching purpose)

武汉大学移动端Totem数据库系统

## 任务要求

Task 1

实现对象Union操作

Task 2

实现更新迁移操作

Task 3

实现手机端各app产生的轨迹（百度地图、跑步软件、共享单车、打车等），并进行Union操作以完整的存储个人出行轨迹

所有功能需要用SQL实现进行调用，可利用javacc进行编译，小组成员共四人，能连通3大任务，并进行流畅的展示

# 背景介绍

---

## Android Studio

这是一个针对Android的IDE，专门为Android开发而设计。它于2013年5月16日在Google I / O 2013年度活动期间启动。Android Studio包含所有用于设计，测试，调试和分析应用程序的Android SDK工具。通过查看开发工具和环境，我们可以将其类似于具有ADT插件的eclipse，但正如我在上面提到的以Android为焦点的IDE一样，Android Studio中提供了许多强大的功能，可以促进和提高您的开发效率。

Android Studio的一些功能有：

- 强大的代码编辑(智能编辑，代码重构)
- 丰富的版式编辑器(当您在版式上拖放视图时，它会显示您在Nexus 4，Nexus 7，Nexus 10和许多其他分辨率的所有屏幕上进行预览。)
- 基于Gradle的构建支持
- Maven支持
- 基于模板的向导
- 皮棉工具分析(Android皮棉工具是一种静态代码分析工具，可检查您的Android项目源文件中是否存在潜在的错误，并针对正确性，安全性，性能，可用性，可访问性和国际化性进行优化改进。)

## javacc

Java Compiler Compiler(JAVACC) 是基于Java应用实现的最受欢迎的语法解析生成器，用于生成词法分析器 (lexical analysers) 和语法分析器 (parsers)。它可以通过读取一个javacc编写的语法规则文件 (包含了词法定义，语法定义的 .jjt结尾的文件)，来生成一个java程序，这个java程序就包括了词法分析器和语法分析器。接着就可以用生成的词法分析器和语法分析器来对我们的输入进行判断，判断输入是否符合我们所要求的语法规则。

JavaCC和传统的LEX+YACC组合不同的是，它是属于自顶向下的语法分析器，用它来构建java方面的语法分析非常容易。

自顶向下分析也称为面向目标的分析方法，简单说来，就是从文法的开始符号出发企图推到出和输入单词串完全相匹配的句子，若输入串是给定文法的句子，则必然能推导出来，否则报错。采用自顶向下的分析方法允许更通用的语法 (但是包含左递归的语法除外)。自顶向下的语法分析器还有其他的一些优点，比如：易于调试，可以分析语法中的任何非终结符，可以在语法分析的过程中在语法分析树中上下传值等。

## Totem和代理类

TOTEM是一种基于对象代理模型的数据库管理系统。把数据存储在建表的概念已经快成了固有的常识了，这是关系数据库最基本的概念之一；但是还有其它的一些方法用于组织数据库。在类Unix操作系统上的文件和目录就形成了一种层次数据库的例子。更现代的发展是面向对象的数据库。与传统面向对象模型相比，对象代理模型更具柔软性，操作更灵活，使用也更方便。

所谓“代理”，是指对一个对象（源对象）的所有或部分属性进行一定的切换（switching）操作，如换名、运算等，而衍生出另外一个对象（代理对象）的过程。通过对象的代理，用户可以将不同的对象进行分割组合，形成新的视图形式的对象。

源对象和代理对象都属于各自的类。因而形成了源类和代理类的概念。拥有切换操作和模式的代理对象被聚集在一个代理类中。代理类对源类的“代理”和一般意义上的“继承”是有区别的。代理类更像是源类的视图，不同的

代理对象表现出了源对象扮演的不同角色。一个源类上可以同时定义多个代理类。代理类也不仅限于代理一个源类，它可以通过连接或并操作同时在几个源类上进行代理。代理类上也可以再定义其它的代理类。代理关系是一个有向图的形状。位于代理层次最顶端的类称为基本类，它们可以看作是没有源类的代理类。

在存储模式上，基本类的对象占有存储空间，存储实际数据。数据库中大部分数据都存储在基本类中。代理对象中的代理属性（称为“虚属性”）只拥有模式而没有实际的物理数据元。源对象和代理对象之间由双向指针来联系，并且由切换操作来确定代理对象中虚属性的值。代理对象可以扩展属于自己属性（称为“实属性”），这些扩展属性才真正占有存储空间。

代理类的定义有点类似于关系数据库中视图的定义。在一般的查询操作上，代理类和普通的类完全一样。

---

## TASK-1: 实现对象Union操作

---

SQL的UNION操作符是用来合并两个或多个SELECT语句的结果,UNION 内部的每个 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每个 SELECT 语句中的列的顺序必须相同。

SQL语句的用法

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

值得注意的是 UNION 是默认选取不同的值,所以我們还需要做一步去重的操作。

或者使用UNION ALL(未实现)

---

实现UNION操作需要修改两个文件 [parse.jj](#) 和 [Transaction.java](#)

每一次[parse.jj](#)的修改之后都需要重新编译生成java的class文件

```
javacc parse.jj
javac *.java
```

一个简单的脚本见[recompile.sh](#)

### javacc中需要注意的地方

- TOKEN加一个 `<UNION:"UNION">`
- 开头补一个 `public static final int OPT_UNION= 9;`

### 基本思路,设计union函数

```
String union():
{
```

```

String union_s;
int count;
}
{
<SELECT> count = directselect()
{
    union_s = OPT_UNION + ",";
    union_s += count;
    while(!st.isEmpty()){
        union_s += ",";
        union_s += st.poll();
    }
}
// 后面部分至少出现一次并且可以循环下去的,所以使用了 +
(<UNION> <SELECT> count = directselect()
{
    union_s += ",";
    union_s += count;
    while(!st.isEmpty()){
        union_s += ",";
        union_s += st.poll();
    }
})+
<SEMICOLON>
{return union_s;}
}

```

但是这样会有一个问题,在sql中需要判断是选择那个语句

```

String[] sql() :
{
    String sql_s;
    String create_s;
    String drop_s;
    String select_s;
    String insert_s;
    String delete_s;
    String update_s;
    String union_s;
}
{
    create_s = create() {sql_s = create_s;System.out.println(sql_s+"\n");return
sql_s.split(","); }
    |drop_s = drop() {sql_s = drop_s;System.out.println(sql_s+"\n");return
sql_s.split(","); }
    |select_s = select(){sql_s = select_s;System.out.println(sql_s+"\n");return
sql_s.split(","); }
    |insert_s = insert2(){sql_s = insert_s;System.out.println(sql_s+"\n");return
sql_s.split(","); }
    |delete_s = delete(){sql_s = delete_s;System.out.println(sql_s+"\n");return
sql_s.split(","); }
    |update_s = update() {sql_s = update_s;System.out.println(sql_s+"\n");return

```

```

sql_s.split(","); }
    |union_s = union() {sql_s = union_s;System.out.println(sql_s+"/n");return
sql_s.split(","); }
}

```

但是 **union** 和 **select** 具有相同的前缀,即左公因子,无法通过LOOKAHEAD来判断应该走哪个

```

options{
    LOOKAHEAD = 3;
    STATIC = false ;
    DEBUG_PARSER = true;
}

```

options中LOOKAHEAD是3,相当于LL(3)文法,而 directinsert() 输入的字符串个数不确定,倒是可以通过改写 LOOKAHEAD = 12来实现对于单个基本SELECT的无限递归查询,但是由于SELECT中可以使用多个嵌套,AS这种无限数量的,所以增大 LOOKAHEAD是治标不治本,并没有实际解决这个问题

## 最终解决方案

修改SELECT,相当于消除左公因子,将其后操作合并为union()函数中解决

```

String select() :
{
    String select_s;
    int count;
    int union_count = 0;
    String union_s; // 用于判断是否是union
}
{
    (<SELECT> count = directselect() union_s = union())
    {
        if (union_s == "END"){
            select_s = OPT_SELECT_DERECTSELECT+", ";
            select_s += count;
            while(!st.isEmpty()){
                select_s += ",";
                select_s += st.poll();
            }
        }
        else {
            select_s = OPT_UNION+", ";
            select_s += count;
            while(!st.isEmpty()){
                select_s += ",";
                select_s += st.poll();
            }
            select_s += union_s;
        }
    }
}

```

```

        return select_s;
    }

    |
    (<SELECT> count = indirectselect() <SEMICOLON>)
    {
        select_s = OPT_SELECT_INDERECTSELECT+",";
        select_s += count;
        while(!st.isEmpty())
        {
            select_s += ",";
            select_s += st.poll();

        }
        return select_s;
    }
}

```

修改后的union函数,用于判断 ; 和 (UNION SELECT)+

如果union\_s返回值是"END"那么就说明使用的是SELECT语句,否则就是UNION语句

```

String union():
{
    String union_s = "";
    int count;
}
{
    <SEMICOLON>
    {
        union_s = "END";
        return union_s;
    }
    |
    // 后面部分至少出现一次并且可以循环下去的,所以使用了 +
    (<UNION> <SELECT> count = directselect()
    {
        union_s += count;
        while(!st.isEmpty()){
            union_s += ",";
            union_s += st.poll();
        }
    })+
    <SEMICOLON>
    {return union_s;}
}

```

## 实现函数执行

这样我们就可以成功解析UNION的语句了,接下来我们需要为UNION完成函数实现,修改[Transaction.java](#)

```
// query选择时补充上UNION的分支
case parse.OPT_UNION:
    log.WriteLog(s);
    Union(aa);
    //new AlertDialog.Builder(context).setTitle("提示").setMessage("合并成功").setPositiveButton("确定",null).show();
    break;
```

接下来是UNION的操作,其实思路比较明确,就是输入是一个 `String []p` 的一个列表,我们需要分析它.

这一步完全可以借鉴SELECT的做法(源文件的SELECT语句有问题,我已经向老师代码仓库提了PR,通过了但是一直没merge我很奇怪??)

代码段我就不贴了, `union()` 函数就是,有点长

主要区别是做一个去重的操作,因为UNION本身是需要去重的,所以复制了一份 `PrintSelectResult` 用于重载

似乎java并不支持默认参数,类似c++ `void f(int a,int b,int c = 10)` 这种写法,所以只能多构造一个参数的

其中的 `removeDuplicate` 这个键用于判断是否去重

```
private void PrintSelectResult(TupleList tpl, String[] attrname, int[] attrid,
String[] type) {
    Intent intent = new Intent(context, PrintResult.class);
    //System.out.println("PrintSelectResult");

    Bundle bundle = new Bundle();
    bundle.putSerializable("tupleList", tpl);
    bundle.putStringArray("attrname", attrname);
    bundle.putIntArray("attrid", attrid);
    bundle.putStringArray("type", type);
    bundle.putString("removeDuplicate", "false");
    intent.putExtras(bundle);
    context.startActivity(intent);
}

private void PrintSelectResult(TupleList tpl, String[] attrname, int[] attrid,
String[] type,String removeDuplicate) {
    Intent intent = new Intent(context, PrintResult.class);
    //System.out.println("PrintSelectResult");

    Bundle bundle = new Bundle();
    bundle.putSerializable("tupleList", tpl);
    bundle.putStringArray("attrname", attrname);
    bundle.putIntArray("attrid", attrid);
    bundle.putStringArray("type", type);
    bundle.putString("removeDuplicate", "true");
    intent.putExtras(bundle);
}
```

```
context.startActivity(intent);  
}
```

## 去重操作

接下来完善PrintResult.java

主要改进思路就是如果removeDuplicate是"true"就去重,采用了一个比较笨的方法判断是否重复

如果是"false"那就还是走原来的路线

值得一提的是java语言判断字符串是否相等使用 `a.equals(b)` 的方式,而不是 `a==b`

java写的不是很熟,比较讨厌这门语言

```
...  
if (removeDuplicate.equals("true")) {  
    String [][]record_list = new String [tabH][tabCol];  
    System.out.println("removeDuplicate");  
    for (int i = 0; i < tabCol; i++) {  
        record_list[0][i] = (tpl.tuplelist.get(0).tuple[attrid[i]]).toString();  
    }  
    int index = 1; // 去重之后的数组长度  
  
    for(int i=1;i<tabH;i++){  
        String []temp_item = new String[tabCol];  
        for(int j=0;j<tabCol;j++){  
            temp_item[j] = (tpl.tuplelist.get(i).tuple[attrid[j]]).toString();  
        }  
        if(!isDuplicate(record_list,temp_item,index)){  
            for(int j=0;j<tabCol;j++){  
                record_list[index][j] = temp_item[j];  
            }  
            index++;  
        }  
    }  
    ...  
}
```

## 测试执行结果

首先创建两个表

需要一行一行执行,这个代码并没有做多行处理

```
CREATE CLASS company1 (name char,age int, salary int);  
INSERT INTO company1 VALUES ("aa",20,1000);  
INSERT INTO company1 VALUES ("bb",30,8000);  
INSERT INTO company1 VALUES ("cc",30,8000);  
INSERT INTO company1 VALUES ("dd",20,1000);
```



```
CREATE CLASS company2 (name char, age int, salary int);
INSERT INTO company2 VALUES ("aa", 20, 1000);
INSERT INTO company2 VALUES ("dd", 30, 1000);
```

### 合并操作

```
SELECT name AS n FROM company1 WHERE age=20 UNION SELECT name AS n FROM company2
WHERE age=30;
```

结果: 执行正确

Print Result	
n	
aa	
dd	

---

## TASK-2: 更新迁移

---

关于更新迁移的定义可以参考[TotemDB资料1](#)中的更新迁移

不过那个文档似乎有些过于啰嗦,对于我这种不太了解数据库也不想去深入了解的看起来实在是有点累,不懂的地方太多..

我简单用语言解释一下:

- 首先我现在CREAT创建了两个class类,这就是两个数据库中的表,我们可以进行增删改查(CRUD).
- 其次我又创建了一个UNION的类,用于合并这两个class表
- 虽然现在没什么问题,但是如果说我以后又在其中某一个class中添加了元素,那么我就需要同步这个UNION的类,这样才能保持同步
- 这显然很麻烦,所以我们需要一个特殊的类,只要某一个class改变了(CRUD),那么我这个UNION的类也会同步改变,这样就可以直接完成同步操作.
- 这个特殊的类就是代理类

代理类有很多种: SELECT 型代理、UNION 型代理、JOIN 型代理、GROUP 型代理. 我们需要实现的就是UNION的代理类

---

### 基本思路

显然这个要求听起来很好,但是实际实现起来其实有点麻烦

但很好的一点是助教完成了SELECT的代理类SELECTDEPUTY,所以我们只需要照葫芦画瓢,仿照SELECTDEPUTY来实现即可

基本思路为：

首先阅读代码后得知，本移动数据库中有以下五个表

**ObjectTable**:存放元组信息（元组名，类名，存储位置）

**SwtchingTable**:存放源类和代理类属性之间的转换关系（例如将源类中元素+2后存到代理类中）

**DeputyTable**：存放代理类与源类的对应关系与代理属性条件

**BiPointerTable**:存放元组同时属于的代理类与源类，实现源类与代理类的对应

**ClassTable**：存放所有类（以属性为单位进行存储）及其信息

而本任务，实现UNION代理类的更新迁移，大致思路则为利用parser解析sql语言生成的信息。

对以上五个表进行修改，将需要代理的源类与属性的信息输入到对应表中。

利用已经实现的更新迁移，实现代理的源类对UNION代理类的更新迁移。

javacc中需要注意的地方

- TOKEN加一个 `<UNIONDEPUTY:"UNIONDEPUTY">`
- 开头补一个 `public static final int OPT_CREATE_UNIONDEPUTY= 10;`

实现语法分析

create里补充一个或判断,用于创建uniondeputy

其中实现CreateUnionDeputy时，为了实现多个UNION使用了+闭包（大于等于一个），实现了任意数量的UNION。

```
String create() :
{
    String create_s;
    int count;
}
{
    ...

    |
    (<CREATE> count = uniondeputy() <SEMICOLON>)
    {
        create_s = OPT_CREATE_UNIONDEPUTY+", ";
        create_s += count;
        while(!st.isEmpty())
        {
            create_s += ",";
            create_s += st.poll();
        }
        return create_s;
    }
}
```

```

int uniondeputy() :
{
    String cln;
    int count;
}
{
    <UNIONDEPUTY> cln = classname() {st.add(cln); }
    <SELECT> count = directselect()
    (<UNION> <SELECT> count = directselect())+ // + reprensts more than one
    {return count;}
}

```

### 修改Transaction.java --- CreateUnionDeputy

通过阅读代码，我们得知更新迁移实现的接口被用在了Transaction.java文件中。

而前人已经实现了SELECT代理类的更新迁移。

我们分析了该更新迁移的外层实现。

SELECT代理类的实现步骤：

1. 解析aa字符串：语句代号，属性数量，代理类名，属性信息（每一个属性占四个位置：代理类属性名，代数变换符号，代数变换数值，源类属性名），源类名，判断条件（三个位置，左值，判断符号，右值），从而初始化代理类类名，代理类类id，代理类属性，源类信息等。
2. 由于储存类信息的表（`classTable`）是以属性为单位，所以对代理类属性（使用源类属性名进行检索），在`classTable`中逐条进行匹配，匹配到属性名相同，则进行下述操作。
3. 在`classTable`中创建条目，属性为代理类信息。
4. 检查属性是否有变换（例如需要+2），同时将变换信息写入`switchingTable`
5. 利用parser解析的条件信息，构建代理类对源类的代理条件，并将代理条件写入`deputyTable`（如age = 20）
6. 在`classTable`中检索condition左值中的元素，获得其id与数据类型。
7. 在`objectTable`中进行匹配，找到代理id与类相同，且满足条件的tuple，录入`objectTable`（录入代理类）
8. 录入时检查是否需要代数switch，若需要，则应该修改值后录入。

我们可以知道，SELECT代理类的功能，是构建一个对一个源类的属性进行变换后代理的代理类。

而我们要实现的UNION代理类是构建一个对多个源类的属性进行变换后代理的代理类，区别仅在于数量。

在上面完成`uniondeputy`的javacc parser后，我们可以得出CreateUnionDeputy需要更改的地方：

1. 我们完成的parser代码生成的aa字符串为：语句代号，属性数量（每一个代理的类属性数量都一样），代理类名，源类1（属性信息（每一个属性占四个位置：代理类属性名，代数变换符号，代数变换数值，源类属性名），源类名，判断条件（三个位置，左值，判断符号，右值）），源类2（同上）.....。
2. 可以看到每一个需要代理的源类由于属性数量相同，所以源类在aa字符串中占用的长度也是相同的，所以我们可以通过加上  $(n-1) * \text{offset}$ ，而这个offset=每一个源类信息的长度，即  $(4 * \text{count} + 4)$  来直接获取到第n个源类的信息。

3. 创建的Union代理类必然会代理两个及以上的源类，所以需要执行SELECTDEPUTY代码中的2~8步大于等于两次。而由于我们防止重复创建，在第二次及以后不需要执行第3，4步，所以将相关代码注释掉（源代码中可以看到）
4. 由于第二次往后的特殊性，我们将该函数代码划分为两部分，第一部分为执行代理类对第一个源类的代理，而第二部分为使用循环，利用offset，对后续每一个源类的进行迭代代理。区别为第一部分会将类属性写入classTable而第二部分不会。

在完成上述表的写入后，系统在每一次数据增加后，都会执行deputy的检查，将源类满足条件但未加入代理类的tuple加入代理类。

## 测试执行结果

先创建两个类

```
CREATE CLASS company1 (name char,age int, salary int);
INSERT INTO company1 VALUES ("aa",20,1000);
INSERT INTO company1 VALUES ("bb",30,8000);

CREATE CLASS company2 (name char,age int, salary int);
INSERT INTO company2 VALUES ("cc",20,1000);
INSERT INTO company2 VALUES ("dd",30,1000);
```

创建UNION代理类

```
CREATE UNIONDEPUTY ud2 SELECT name AS n,age AS a FROM company1 WHERE salary=1000
UNION SELECT name AS n,age AS a FROM company2 WHERE salary=1000;
```

提示union代理类创建成功

提示

创建Union代理类成功

确定

查询代理类中情况

```
SELECT n AS n1 FROM ud2 WHERE a=20;
```

结果: 执行正确

Print Result	
n1	
aa	
cc	

UNION代理类代理多个对象

```
CREATE CLASS company3 (name char,age int, salary int);
INSERT INTO company3 VALUES ("ee",20,1000);
INSERT INTO company4 VALUES ("ff",30,1000);

CREATE UNIONDEPUTY ud3 SELECT name AS n,age AS a FROM company1 WHERE salary=1000
UNION SELECT name AS n,age AS a FROM company2 WHERE salary=1000 UNION SELECT name
AS n,age AS a FROM company3 WHERE salary=1000;
```

测试

```
SELECT n AS n1 FROM ud3 WHERE a=20;
```

Print Result	
n1	
aa	
cc	
ee	

```
INSERT INTO company1 VALUES ("zz",20,1000);
SELECT n AS n1 FROM ud2 WHERE a=20;
SELECT n AS n1 FROM ud3 WHERE a=20;
```

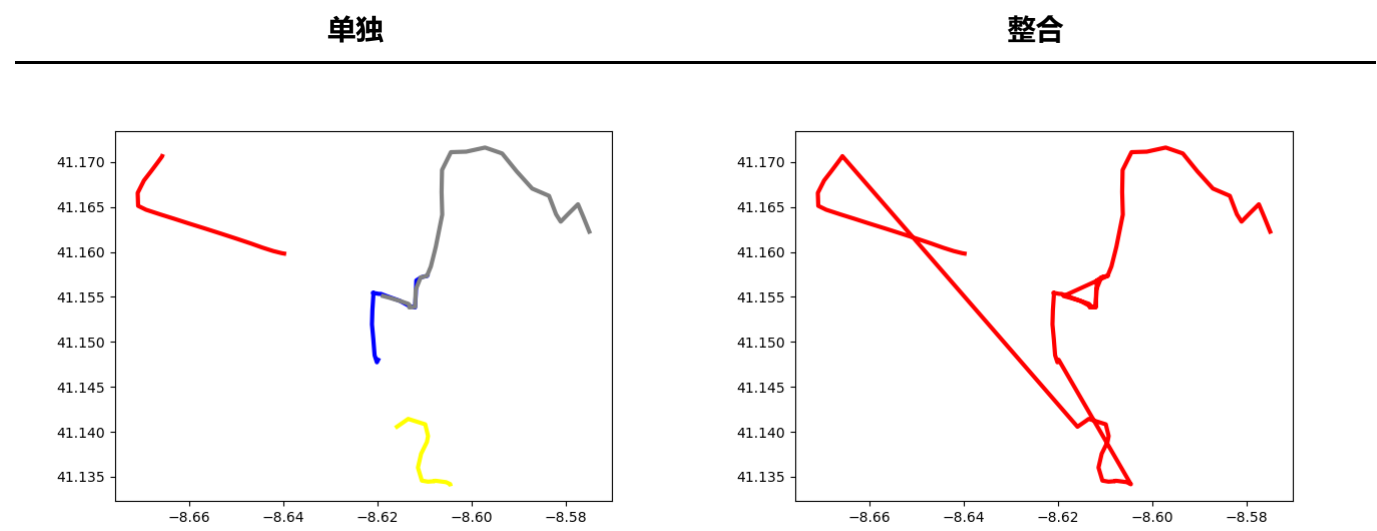
ud2	ud3

ud2	ud3
<div>Print Result</div> <div>n1</div> <div>aa</div> <div>cc</div> <div>zz</div>	<div>Print Result</div> <div>n1</div> <div>aa</div> <div>cc</div> <div>ee</div> <div>zz</div>

## TASK-3 app数据轨迹绘制

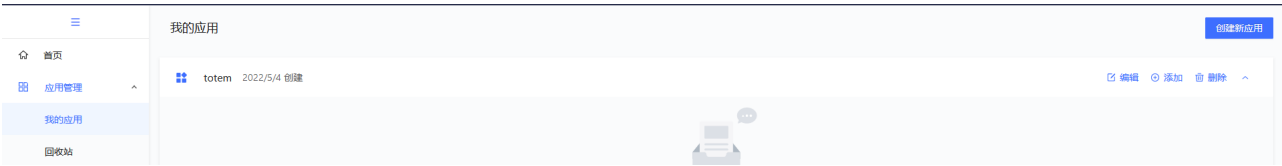
python绘图脚本 -finished by xuzijun

所有的app的数据放在dataset文件夹下,一共有四组app的部分数据,使用python绘制出来的结果是



其中整合操作其实就是union的操作,合并不同app的数据,将结果

- 1. 进入高德开放平台,注册
- 2. 我的应用,创建一个新的应用



- 3. 获取安卓密钥

将luzhi改为你的电脑的用户名

默认的保存位置在 C:/Users/luzhi/.android,进入这个目录,输入

```
keytool -list -v -keystore C:/Users/luzhi/.android/debug.keystore
```

默认口令是 **android**

```
C:\Users\luzhi\.android>keytool -list -v -keystore C:\Users\luzhi\.android\debug.keystore
输入密钥库口令:
密钥库类型: PKCS12
密钥库提供方: SunJSSE

您的密钥库包含 1 个条目

别名: androiddebugkey
创建日期: 2022-4-7
条目类型: PrivateKeyEntry
证书链长度: 1
证书[1]:
所有者: C=US, O=Android, CN=Android Debug
发布者: C=US, O=Android, CN=Android Debug
序列号: 1
生效时间: Thu Apr 07 10:07:24 CST 2022, 失效时间: Sat Mar 30 10:07:24 CST 2052
证书指纹:
    SHA1: [redacted]
    SHA256: [redacted]
签名算法名称: SHA1withRSA (弱)
主体公共密钥算法: 2048 位 RSA 密钥
版本: 1

*****
*****
```

记录其中的SHA1

#### 4. 回到我的应用,添加

其中包名是build.gradle中的applicationId,本项目是**drz.oddb**

+

为「totem」添加Key

X

\* Key名称:

totem

命名规范

\* 服务平台:

Android平台

iOS平台

Web端(JS API)

Web服务

智能硬件

微信小程序

HarmonyOS平台

可使用服务:

Android地图SDK

Android定位SDK

Android导航SDK

Android室内地图SDK

Android室内定位SDK

Android猎鹰SDK

\* 发布版安全码SHA1:

如何获取

调试版安全码SHA1:

请输入调试版安全码SHA1

\* PackageName:

drz.oddb

如何获取

阅读并同意

高德服务条款及隐私权政策、高德地图开放平台服务协议和高德地图开放平台隐私权政策

为了尽可能降低您的违规风险,请您务必在隐私政策中说明SDK提供者的公司名称、SDK名称、使用目的、收集和使用的个人信息类型以及透出并可访问《高德地图开放平台隐私权政策》的访问链接即<https://lbs.amap.com/home/privacy/>。

取消

提交

这会给你分配一个key值,比如我的是4ff28d19e341ebf29b6667d56435c5d2

5. 进入高德地图的 Android 地图SDK下载

下滑,选择开发包定制下载



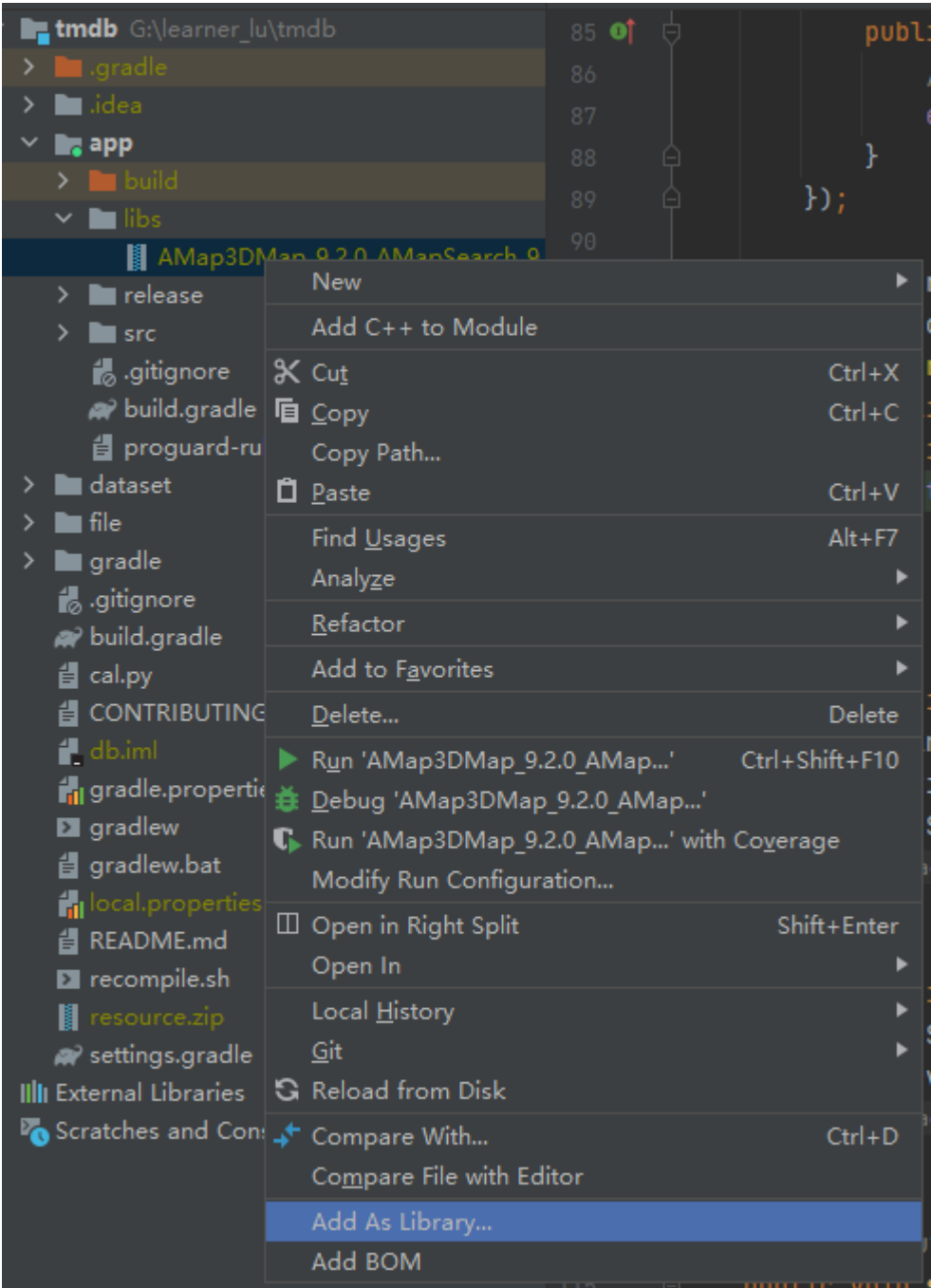


6. 解压后得到如下文件

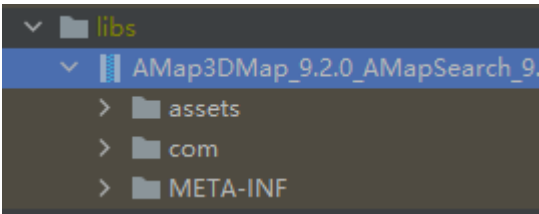
- arm64-v8a
- armeabi
- armeabi-v7a
- x86
- x86\_64
- AMap3DMap\_9.2.0\_AMapSearch\_9.2.0\_20220414.jar

在 app下新建文件夹,命名为 lib,将最后一个文件复制到该文件夹下

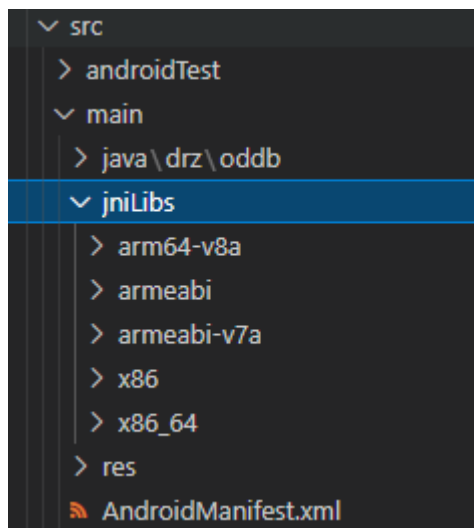
在Android studio下点击该包,选择Add as library



接下来这个包会被展开,得到如下的目录



- 7. 在app/src/main下新建文件夹,命名为 jniLibs,将刚才解压的SDK那五个文件夹复制到这个目录下面,最后得到如下目录结构



8. 将如下内容复制到AndroidManifest.xml,复制在<manifest>之间</manifest>

```
<!--允许程序打开网络套接字-->
<uses-permission android:name="android.permission.INTERNET" />
<!--允许程序设置内置sd卡的写权限-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<!--允许程序获取网络状态-->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<!--允许程序访问WiFi网络信息-->
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!--允许程序读写手机状态和身份-->
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<!--用于进行网络定位-->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION">
</uses-permission>
<!--用于访问GPS定位-->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION">
</uses-permission>
<!--用于获取wifi的获取权限，wifi信息会用来进行网络定位-->
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"></uses-
permission>
<!--用于读取手机当前的状态-->
<uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-
permission>
<!--用于申请调用A-GPS模块-->
<uses-permission
android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS"></uses-
permission>
```

然后在 <application></application>中间添加代码

将后面的value改为你的key值,当然你如果用我的也是可以的~

```
<meta-data android:name="com.amap.api.v2.apikey"
android:value="4ff28d19e341ebf29b6667d56435c5d2">
</meta-data>
```

## 9. 接下来是代码的编写.

我们首先分析一下需求,要求的是可以追踪轨迹,并且可以UNION所有APP的数据进行追踪.说实话这个要求有点难,所有的app数据放在dataset里,我们需要把他们先都导入数据库,然后选择性的建立UNION代理类,然后输出每一个UNION代理类的结果,追踪轨迹

这里有几个问题,首先是导入. 助教这个代码怎么说呢,只实现了int和char的,没有实现float和double的这个要自己写一下

其次是选择UNION代理类输出轨迹,这个需要一个比较高级的安卓窗口,一些选项按钮啥的

所以我的选择是直接就是把数据搞出来,直接写死了,就当作直接UNION了所有APP的数据,并且不会更新.事实上这种做法相当的愚蠢

好,那现在问题就锁定到了,如何实现轨迹追踪?

## 10. 这里我们就需要查阅高德地图的文档了和安卓的文档了

可能这里我说的话很轻松,不过我相信查过文档写代码的都知道,这种有多费劲,有多难debug

我们首先创建一个新的界面(视图),在app/src/main/res/layout 下新建一个文件,命名为gaodemap.xml,用于生成高德地图的界面,复制以下内容

```
<com.amap.api.maps.MapView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mapView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
</com.amap.api.maps.MapView>
```

其中xmlns:android是提供一个作用域,类似namespace,android:id是用来给一个名字,可以通过id找到这个视图,用于切换页面显示页面

## 11. 在app/src/main/java/drz/oddb 下创建一个文件,命名为gaodemap.java,用于实现功能

其基础功能如下所示,需要继承安卓的AppCompatActivity类并重写(override)其方法

AMap和MapView是高德地图包的类名,我们需要使用其创建视图

java中class名与文件名一致,采用的是包的管理策略

```
import com.amap.api.maps.AMap;
import com.amap.api.maps.MapView;
import android.support.v7.app.AppCompatActivity;

public class gaodemap extends AppCompatActivity{

    private AMap aMap;
```

```
MapView mMapView = null;

@Override
protected void onDestroy() {
    super.onDestroy();
    //在activity执行onDestroy时执行mMapView.onDestroy(), 销毁地图
    mMapView.onDestroy();
}

@Override
protected void onResume() {
    super.onResume();
    //在activity执行onResume时执行mMapView.onResume (), 重新绘制加载地图
    mMapView.onResume();
}

@Override
protected void onPause() {
    super.onPause();
    //在activity执行onPause时执行mMapView.onPause (), 暂停地图的绘制
    mMapView.onPause();
}

@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    //在activity执行onSaveInstanceState时执行mMapView.onSaveInstanceState
    (outState), 保存地图当前的状态
    mMapView.onSaveInstanceState(outState);
}
```

重点是创建onCreate函数,用于生成视图

```
...
import com.amap.api.services.core.ServiceSettings;

@Override
protected void onCreate() {
    super.onCreate(savedInstanceState);

    // 询问隐私政策
    ServiceSettings.updatePrivacyShow(this, true, true);
    ServiceSettings.updatePrivacyAgree(this,true);

    setContentView(R.layout.gaodemap);
    mMapView = (MapView)findViewById(R.id.mapView);
    //在activity执行onCreate时执行mMapView.onCreate(savedInstanceState), 创建
    地图
    mMapView.onCreate(savedInstanceState);

    //初始化地图控制器对象
    if (aMap == null) {
        aMap = mMapView.getMap();
    }
}
```

值得一提的是,高德地图在2021年11月之后出台了[隐私政策](#),需要明确向用户提出授权隐私政策

```
ServiceSettings.updatePrivacyShow(this, true, true);
ServiceSettings.updatePrivacyAgree(this, true);
```

这两行不加是不行的

然后我们需要在AndroidManifest.xml的<application>之间</application>注册一个事件,用于启动该事件,我这里就命名为gaodemap

```
<activity android:name=".gaodemap">
    <intent-filter>
        <action android:name="android.intent.action.gaodemap" />

        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

12. 现在高德地图的画面部分就已经完成了.我们需要一个启动方式,所以就在主界面里加一些按钮,然后把按钮的事件绑定到函数即可

activity\_main.xml中修改增加两个按钮(button)

分别命名为clean\_button和draw\_trace,第一个用于清除文本框中的文字,这个主要是因为以前调试的时候需要输入,删除,删除还需要长按全选然后再删除,本着省时省力的原则设计了这个按钮.

第二个按钮就是跳转到轨迹追踪的界面.这两个按钮可以在右侧的background属性中选择一个合适的图片,也可以自定义目录位置如@drawable/xxx使用一张图片

然后在主函数MainActivity.java中添加两个方法

```
// 清除文本框内数据
Button clean_button = findViewById(R.id.clean_button);
clean_button.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v){
        //editText = findViewById(R.id.edit_text);
        editText.setText("");
    }
});
Button draw_trace = findViewById(R.id.draw_trace);
draw_trace.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v){
        trans.show_map();
    }
});
```

```
    }  
});
```

注意到这里的`R.id.clean_button`和`R.id.draw_trace`其实就是主视图中的按钮名

然后在`Transaction.java`中添加`show_map`函数,就是启动这个事件

```
// 不要忘记import包  
import drz.oddb.gaodemap;  
  
public void show_map(boolean whu){  
    Intent intent = new Intent(context, gaodemap.class);  
    context.startActivity(intent);  
}
```

接下来运行程序点击按钮就可以看到一个高德地图的画面了,默认定位地点在北京

由于我是事后补充的这个文档,可能有的地方写的不是很详细,也可能会遇到一些其他的问题,可以参考这次[commit](#)前后的对比,这里的画像是xzj做的

### 13. 接着就是绘图了

这一部分应该说是最麻烦的,不过我也没什么好说的,查文档查到的,就是一个点平滑移动+颜色,具体内容看代码吧,注释还是比较清晰的

值得一提的是,java有时候需要import外部的包,但是我并不知道包名叫什么,文档里也没有个提示,这属实是很令人头大

但是有一个好消息是你可以把代码复制到Android studio里,没有引用的函数/类他都会自动检测,直接帮你import了,这可是帮了我大忙.

### 14. 最后一部分就是我想加的了,就是记录一下我的生活轨迹,从梅六走到计算机学院的路线再走回来

你可以在这个[网站](#)上查询到精确的经纬度坐标,不过似乎有一点微小的偏差

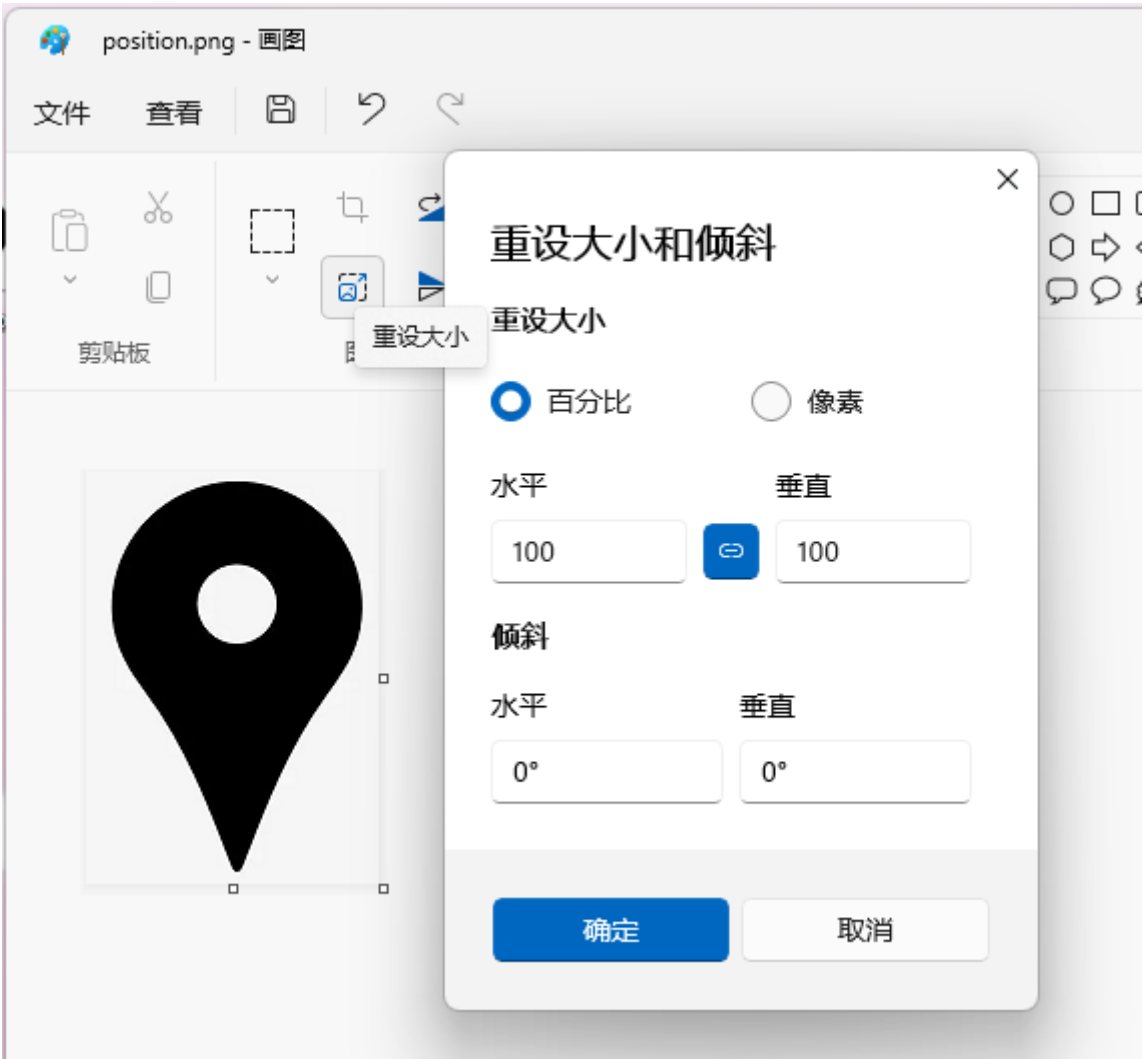
另外值得一提的是`LatLng`的属性是先纬度后经度,这个如果写反了的话是不行的.会显示不出来地图.

其他国家的地图并不会显示精确的地理信息,只有中国的地图上才会有很精确的细节,路/宿舍都会有标注(为啥没有梅六???)

然后又补充了一个武大的校徽,然后做了一个判断.使用的`bundle`和`intent`来封装传参,可以显示APP的也可以显示我的路线,这个应该算是照葫芦画瓢吧哈哈

### 15. 最后的话就是给点搞一个图片,显得专业一点,如果图片太大的话效果很不好

可以直接使用画图重设分辨率大小,不需要PS



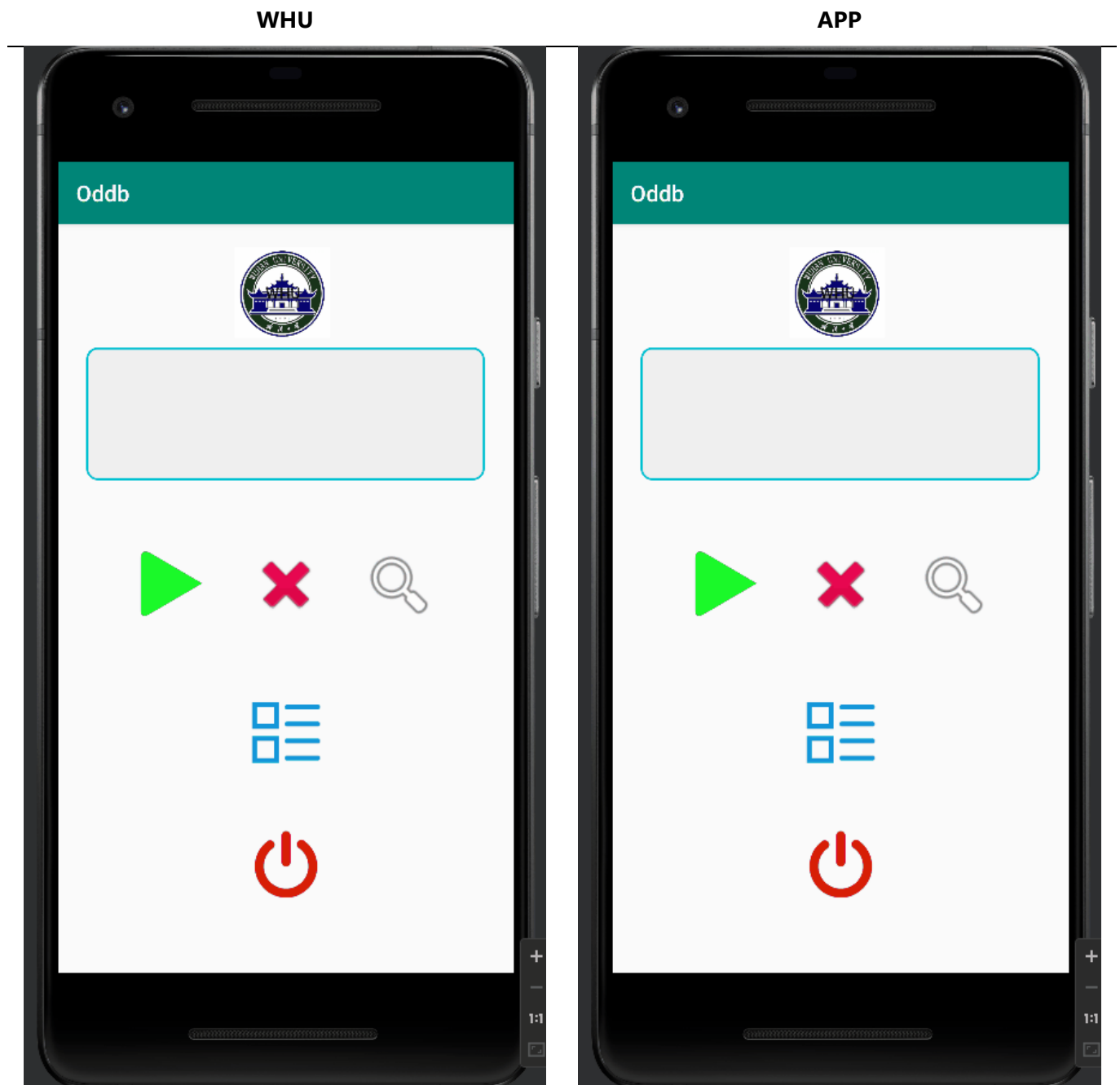
最后的话,非常感谢你能有耐心看到这里,写文档不易,看内容也不易,祝好运~

结果演示

WHU

APP





## 参考资料

<https://lbs.amap.com/api/android-sdk/guide/create-project/android-studio-create-project>

[https://blog.csdn.net/qq\\_50272406/article/details/123006575](https://blog.csdn.net/qq_50272406/article/details/123006575)

## 一些锦上添花

### 音乐

调用音乐播放器的位置是[MusicServer.java](#),其引用了Android库中MediaPlayer组件

最开始播放的音乐是[R.raw.old\\_memory](#),这是缘之空的音乐很好听.


现在是无法直接播放的,因为没有后缀windows无法识别文件格式. 添加后缀.mp4即可播放

如果你想替换这个背景音乐,那么直接下载一个音乐,去掉后缀,替换文件的值即可

我选的这个是EVA的一首歌 one last kiss

给项目提PR

fix bug for select SQL #1

 Open


luzhixing12345 wants to merge 1 commit into `whu-totemdb:main` from `learner-lu:main`

Conversation 0

Commits 1



Checks 0


Files changed 2




luzhixing12345 commented 19 hours ago

No description provided.


  fix bug for select SQL 7214a94



 tgbnhy approved these changes 12 hours ago

[View changes](#)


Add more commits by pushing to the `main` branch on `learner-lu/tmdb`.



Changes approved

1 approving review [Learn more](#).

1 approval



This branch has no conflicts with the base branch

Only those with [write access](#) to this repository can merge pull requests.

如果你尚不熟悉如果给一个项目提 pull request(PR),可以参考我的博客-[如何给开源项目提PR](#)

替换图标

- 1. 制作一个图标 (png/jpg),放在(app/src/main/res/drawable)文件夹下

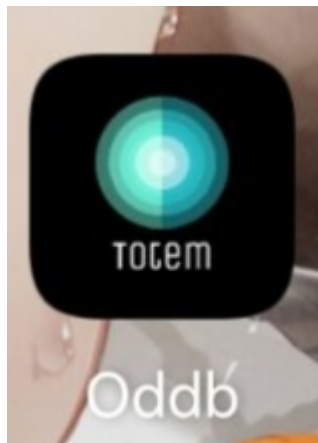


2. 修改AndroidManifest.xml中的 `android:icon="@drawable/xxx"`,不需要png/jpg后缀

3. 导出apk

此步可能会出现一些问题,如果报错Lint found fatal ...,可以参考[博客](#),修改[build.gradle](#)

4. wow, 还不错哦~



支持大小写SQL语句

这里值得一提的是,因为ID和STRING都是使用的正则处理,所以需要提前一些小写字符,不然会被优先识别为ID

```
TOKEN:
{
<SEMICOLON: "; ">
|<CREATE: "CREATE" | "create">
|<DROP: "DROP" | "drop">
|<CLASS: "CLASS" | "class">
|<INSERT: "INSERT" | "insert">
|<INTO: "INTO" | "into">
|<VALUES: "VALUES" | "values">
|<LEFT_BRACKET: "(">
|<COMMA: ", ">
|<RIGHT_BRACKET: ")">
|<DELETE: "DELETE">
|<FROM: "FROM" | "from">
|<WHERE: "WHERE" | "where">
|<SELECT: "SELECT" | "select">
|<SELECTDEPUTY: "SELECTDEPUTY" | "selectdeputy">
|<UNIONDEPUTY: "UNIONDEPUTY" | "uniondeputy">
|<AS: "AS" | "as">
|<UPDATE: "UPDATE" | "undate">
|<SET: "SET" | "set">
|<UNION: "UNION" | "union">
|<ID: ["a"-"z"](["a"-"z", "A"-"Z", "0"-"9"])* >
|<EQUAL: "=">
|<INT: "0" | (["1"-"9"](["0"-"9"])* ) >
|<STRING: "\""(["a"-"z", "A"-"Z", "1"-"9"])* "\"" >
|<CROSS: "->">
|<DOT: ".">
|<PLUS: "+">
}
```