测试

- 黑盒测试
  - 测试逻辑业务
- 白盒测试
  - 测试逻辑方法
- 根据测试粒度
  - 方法测试：function test
  - 单元测试：unit test
  - 集成测试：integration test
  - 系统测试：system test
- 根据测试暴力程度
  - 冒烟测试：smoke test
  - 压力测试：pressure test

单元测试junit

- 定义一个类继承AndroidTestCase，在类中定义方法，即可测试该方法
- 在指定指令集时，targetPackage指定你要测试的应用的包名

```
<instrumentation
android:name="android.test.InstrumentationTestRunner"
android:targetPackage="com.itheima.junit"
></instrumentation>
```

- 定义使用的类库

```
<uses-library android:name="android.test.runner"></uses-library>
```

- 断言的作用，检测运行结果和预期是否一致
- 如果应用出现异常，会抛给测试框架

SQLite数据库

- 轻量级关系型数据库
- 创建数据库需要使用的api：SQLiteOpenHelper
  - 必须定义一个构造方法：

```
//arg1:数据库文件的名字
//arg2:游标工厂
//arg3:数据库版本
public MyOpenHelper(Context context, String name, CursorFactory factory, int version){}
```

  - 数据库被创建时会调用：onCreate方法
  - 数据库升级时会调用：onUpgrade方法

创建数据库

```
//创建OpenHelper对象
MyOpenHelper oh = new MyOpenHelper(getContext(), "person.db", null, 1);
//获得数据库对象,如果数据库不存在，先创建数据库，后获得，如果存在，则直接获得
```

```
SQLiteDatabase db = oh.getWritableDatabase();
```

- getWritableDatabase()：打开可读写的数据库
- getReadableDatabase()：在磁盘空间不足时打开只读数据库，否则打开可读写数据库
- 在创建数据库时创建表

```
public void onCreate(SQLiteDatabase db) {
    // TODO Auto-generated method stub
    db.execSQL("create table person (_id integer primary key autoincrement, name char(10), phone char(20), money integer(20))"
}
```

数据库的增删改查

SQL语句

- insert into person (name, phone, money) values ('张三', '159874611', 2000);
- delete from person where name = '李四' and _id = 4;
- update person set money = 6000 where name = '李四';
- select name, phone from person where name = '张三';

执行SQL语句实现增删改查

```
    //插入
    db.execSQL("insert into person (name, phone, money) values (?, ?, ?);", new Object[]{"张三", 15987461, 75000});
    //查找
    Cursor cs = db.rawQuery("select _id, name, money from person where name = ?;", new String[]{"张三"});
```

- 测试方法执行前会调用此方法

```
protected void setUp() throws Exception {
    super.setUp();
    //                  获取虚拟上下文对象
    oh = new MyOpenHelper(getContext(), "people.db", null, 1);
}
```

使用api实现增删改查

- 插入

```
//以键值对的形式保存要存入数据库的数据
ContentValues cv = new ContentValues();
cv.put("name", "刘能");
cv.put("phone", 1651646);
cv.put("money", 3500);
//返回值是改行的主键，如果出错返回-1
long i = db.insert("person", null, cv);
```

- 删除

```
//返回值是删除的行数
int i = db.delete("person", "_id = ? and name = ?", new String[]{"1", "张三"});
```

- 修改

```
ContentValues cv = new ContentValues();
cv.put("money", 25000);
int i = db.update("person", cv, "name = ?", new String[]{"赵四"});
```

- 查询

```
//arg1:要查询的字段
//arg2：查询条件
//arg3:填充查询条件的占位符
Cursor cs = db.query("person", new String[]{"name", "money"}, "name = ?", new String[]{"张三"}, null, null, null);
while(cs.moveToNext()){
    //                          获取指定列的索引值
    String name = cs.getString(cs.getColumnIndex("name"));
    String money = cs.getString(cs.getColumnIndex("money"));
    System.out.println(name + ";" + money);
}
```

事务

- 保证多条SQL语句要么同时成功，要么同时失败
- 最常见案例：银行转账
- 事务api

```
try {
    //开启事务
    db.beginTransaction();
    ...........
    //设置事务执行成功
    db.setTransactionSuccessful();
} finally{
    //关闭事务
    //如果此时已经设置事务执行成功，则sql语句生效，否则不生效
    db.endTransaction();
}
```

把数据库的数据显示至屏幕

1. 任意插入一些数据
2. 定义业务bean：Person.java
3. 读取数据库的所有数据

```
Cursor cs = db.query("person", null, null, null, null, null, null);
while(cs.moveToNext()){
    String name = cs.getString(cs.getColumnIndex("name"));
    String phone = cs.getString(cs.getColumnIndex("phone"));
    String money = cs.getString(cs.getColumnIndex("money"));
    //把读到的数据封装至Person对象
    Person p = new Person(name, phone, money);
    //把person对象保存至集合中
    people.add(p);
}
```

4. 把集合中的数据显示至屏幕

```
 LinearLayout ll = (LinearLayout) findViewById(R.id.ll);
 for(Person p : people){
     //创建TextView，每条数据用一个文本框显示
     TextView tv = new TextView(this);
     tv.setText(p.toString());
     //把文本框设置为ll的子节点
     ll.addView(tv);
 }
```

5. 分页查询

```
Cursor cs = db.query("person", null, null, null, null, null, null, "0, 10");
```

ListView

- 就是用来显示一行一行的条目的
- MVC结构
  - M：model模型层，要显示的数据 ————people集合
  - V：view视图层，用户看到的界面 ————ListView
  - c：control控制层，操作数据如何显示 ————adapter对象
- 每一个条目都是一个View对象

BaseAdapter

- 必须实现的两个方法
  - 第一个

```
//系统调用此方法，用来获知模型层有多少条数据
@Override
public int getCount() {
    return people.size();
}
```

  - 第二个

```
//系统调用此方法，获取要显示至ListView的View对象
//position:是return的View对象所对应的数据在集合中的位置
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    System.out.println("getView方法调用" + position);
    TextView tv = new TextView(MainActivity.this);
    //拿到集合中的元素
    Person p = people.get(position);
    tv.setText(p.toString());

    //把TextView的对象返回出去，它会变成ListView的条目
    return tv;
}
```

- 屏幕上能显示多少个条目，getView方法就会被调用多少次，屏幕向下滑动时，getView会继续被调用，创建更多的View对象显示至屏幕

条目的缓存

- 当条目划出屏幕时，系统会把该条目缓存至内存，当该条目再次进入屏幕，系统在重新调用getView时会把缓存的条目作为convertView参数传入，但是传入的条目不一定是之前被缓存的该条目，即系统有可能在调用getView方法获取第一个条目时，传入任意一个条目的缓存

对话框

确定取消对话框

- 创建对话框构建器对象，类似工厂模式

```
AlertDialog.Builder builder = new Builder(this);
```

- 设置标题和正文

```
builder.setTitle("警告");
builder.setMessage("若练此功，必先自宫");
```

- 设置确定和取消按钮

```
builder.setPositiveButton("现在自宫", new OnClickListener() {

    @Override
    public void onClick(DialogInterface dialog, int which) {
        // TODO Auto-generated method stub
        Toast.makeText(MainActivity.this, "恭喜你自宫成功，现在程序退出", 0).show();
    }
});

builder.setNegativeButton("下次再说", new OnClickListener() {

    @Override
    public void onClick(DialogInterface dialog, int which) {
        // TODO Auto-generated method stub
        Toast.makeText(MainActivity.this, "若不自宫，一定不成功", 0).show();
    }
});
```

- 使用构建器创建出对话框对象

```
AlertDialog ad = builder.create();
ad.show();
```

单选对话框

```
    AlertDialog.Builder builder = new Builder(this);
    builder.setTitle("选择你的性别");
```

- 定义单选选项

```
final String[] items = new String[]{
        "男", "女", "其他"
};
//-1表示没有默认选择
//点击侦听的导包要注意别导错
builder.setSingleChoiceItems(items, -1, new OnClickListener() {

    //which表示点击的是哪一个选项
    @Override
    public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(MainActivity.this, "您选择了" + items[which], 0).show();
        //对话框消失
        dialog.dismiss();
    }
});

builder.show();
```

多选对话框

```
    AlertDialog.Builder builder = new Builder(this);
    builder.setTitle("请选择你认为最帅的人");
```

- 定义多选的选项，因为可以多选，所以需要一个boolean数组来记录哪些选项被选了

```
final String[] items = new String[]{
        "赵帅哥",
        "赵师哥",
        "赵老师",
        "侃哥"
};
//true表示对应位置的选项被选了
```

```
final boolean[] checkedItems = new boolean[]{
        true,
        false,
        false,
        false,
};
builder.setMultiChoiceItems(items, checkedItems, new OnMultiChoiceClickListener() {

    //点击某个选项，如果该选项之前没被选择，那么此时isChecked的值为true
    @Override
    public void onClick(DialogInterface dialog, int which, boolean isChecked) {
        checkedItems[which] = isChecked;
    }
});

builder.setPositiveButton("确定", new OnClickListener() {

    @Override
    public void onClick(DialogInterface dialog, int which) {
        StringBuffer sb = new StringBuffer();
        for(int i = 0;i < items.length; i++){
            sb.append(checkedItems[i] ? items[i] + " " : "");
        }
        Toast.makeText(MainActivity.this, sb.toString(), 0).show();
    }
});
builder.show();
```