项目名称

组长:郑海君
学号 15352437
邮箱：1269418465@qq.com
qq 号:1269418465

成员 1： 赵钰莹
学号：15352433
成员 2：郑晓如
学号：15352440

目录

一、概述

本三国电子词典收录三国演义和三国志中出现的部分人物，详尽提供了多样的三国英雄人物的个人数据。利用数据库技术，方便快捷地对三国人物信息进行数据整理和显示，还提供智能搜索功能，可根据搜索内容快速查找对应英雄人物；提供“增、删、改”功能，允许用户创建或删除人物或对本三国电子词典的人物信息进行修改；同时，本应用还提供英雄人物 PK 对决功能，娱乐性较强。

二、背景

在实现“中国梦”的时代背景下，取传统文化之精华有利于在新时代中升华自我。中华民族历史悠久，中国传统文化博大精深，五千年文明灿烂不熄。其中，三国时期更可谓是洋溢着英雄主义的战争史诗，是一幅充满着智慧和经验的历史画卷。

基于以上的背景，本应用旨在提供三国人物信息查询、增加、删除、修改功能供用户方便快捷地了解三国英雄信息，并通过三国英雄人物 PK 的趣味环节，寓教于乐。

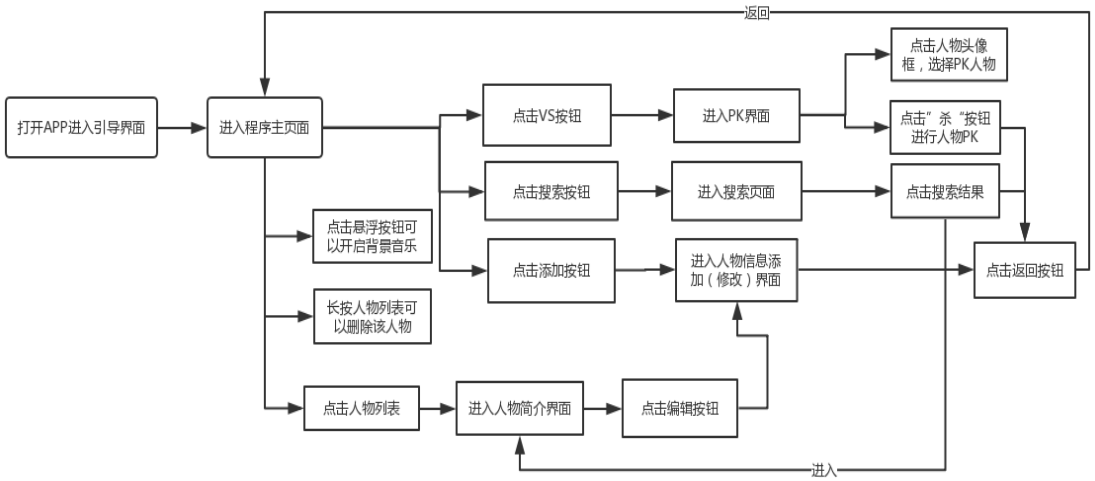
三、项目说明

1、 功能模块

(1) 以 SQLite 数据库为基础的增删查改功能
- 增加英雄人物：可添加新增人物的姓名、籍贯、性别、主效势力、生卒年月、简介以及人物图片。其中人物图片的来源可以是直接用相机拍照获得，也可以从相册中选取；
- 删除英雄人物：长按列表中的人物，会弹出对话框询问是否确定删除人物，如果点击确定，则删除；否则，取消删除；
- 查找英雄人物：在搜索框中输入人物名字，便可查到数据库中对应的人物；
- 修改人物信息：点击编辑按钮，可以对人物的详细信息进行修改，可以更换人物的图片（来源是相机拍取或本地相册）、姓名等信息；

(2) 英雄 PK 功能
- 可以选择自己喜欢的两个英雄任务进行 PK；
- 点击 PK 按钮，两个人物开始 PK，此时，我们可以看到人物 PK 的动画效果，也能观察到武力值低的人物的血槽逐渐变空，还能看到对战的两个人物的对话，对话内容是放在气泡动画的效果中，十分形象逼真。

2、 逻辑流程



3、 技术说明（重点）

(1) 背景音乐的实现

- ✧ 自定义一个类“MusicService”继承自 Service,Service 就是用来在后台完成一些不需要和用户交互的动作的。

```
public class MusicServer extends Service implements MediaPlayer.OnCompletionListener
```

- ✧ 为了和 Activity 进行交互，自定义一个 Binder 对象

```
class AudioBinder extends Binder {
        //返回 Service 对象
        MusicServer getService(){
            return MusicServer.this;
        }
    }
```

- ✧ 重载 onBind 方法，当其他组件调用 bindService()来绑定该 Service 时，系统会调用这个回调函数，这里提供的 Client 和 Service 通信的接口就是自定义的 binder 对象。

```
private final IBinder binder = new AudioBinder();
    @Override
    public final IBinder onBind(Intent intent) {
        return binder;
    }
```

- ✧ 实例化 MediaPlayer 对象，并且从 raw 文件夹中获取一个应用自带的 mp3 文件，setOnCompletionListener(MediaPlayer.OnCompletionListener listener)方法的作用是：当流媒体播放完毕的时候回调。

```
public void onCreate(){
      super.onCreate();
```

```
        mediaPlayer = MediaPlayer.create(this, R.raw.music);
        mediaPlayer.setOnCompletionListener(this);
    }
```

- ✧ 实现 onStartCommand 方法，代替原来的 onStart 方法，如果音乐没有在播放，那么就打开播放器，同时返回 START_STICKY，它的含义是：如果 service 进程被 kill 掉，保留 service 的状态为开始状态，但不保留递送的 intent 对象。随后系统会尝试重新创建 service，由于服务状态为开始状态，所以创建服务后一定会调用 onStartCommand(Intent,int,int)方法。如果在此期间没有任何启动命令被传递到 service，那么参数 Intent 将为 null。

```
public int onStartCommand(Intent intent, int flags, int startId){
        if(!mediaPlayer.isPlaying()){
            mediaPlayer.start();
        }
        return START_STICKY;
    }
```

- ✧ 重写 onDestroy 方法，停止音乐播放，同时释放资源

```
@Override
    public void onDestroy() {
        super.onDestroy();
        mediaPlayer.stop();
        mediaPlayer.release();
    }
```

- ✧ 在应用主界面，当需要启动音乐播放器的时候，调用 bindService 绑定服务，并且执行 startService 启动服务

```
bindService(intent, conn, Context.BIND_AUTO_CREATE);
startService(intent);
```

- ✧ 停止应用播放时，解除绑定，并且停止服务

```
unbindService(conn);
stopService(intent);
```

(2) 自定义 AlertDialog 背景

- ✧ 设计 AlertDialog 的背景图，该界面包含四个控件，两个 TextView 和两个 Button，给主界面加上背景图，为了使该背景图可以透过 Button，可以把 Button 的背景色设置为透明的；

```
android:background="#00000000"
```

- ✧ 在主界面里，实例化 AlertDialog.Builder 对象 builder，用 setView 来加载自定义的布局，同时，通过 findViewById 来找到布局上的对应控件，并且对它们进行具体操作。Builder 只是将 AlterDialog 的参数存放在一个参数对象里，做一个暂时的保存，在最后调用 create 方法的时候，创建一个 AlterDialog 对象，将暂存的参数一次性应用到这个 Dialog 对象上。

```
final AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);
View view = View.inflate(MainActivity.this,R.layout.my_alert_layout,
null);
builder.setView(view);
```

```
builder.setCancelable(true);
final AlertDialog dialog = builder.create();
dialog.show();
```

(3) 带删除键的 EditText 的实现

✧ 首先对 EditText 设置修改监听事件，在 EditText 还未被修改前，把删除按钮设置为不可见的，当检测到 EditText 的内容发生变化，并且结束光标的位置不等于 0 时，删除按钮可见，测试点击删除按钮，就把 EditText 的内容设置为空的。

```
editText.addTextChangedListener(new TextWatcher(){
      @Override
      public void beforeTextChanged(CharSequence s, int start, int count,
int after) {
                imageButton.setVisibility(View.INVISIBLE);
      }
      @Override
       public void afterTextChanged(Editable s){
                editStart = editText.getSelectionStart();
editEnd = editText.getSelectionEnd();
                if(editEnd!=0){
                     imageButton.setVisibility(View.VISIBLE);
                }
          }
 });
```

(4) 自定义圆形 CircleImageView 及其应用

✧ 在 res/values/attrs.xml 中定 CircleImageView 的参数值

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="CircleImageView">
        <attr name="border_width" format="dimension" />
        <attr name="border_color" format="color" />
    </declare-styleable
</resources>
```

✧ 写一个继承自 ImageView 的控件:CircleImageView

✧ 创建构造方法

通过 obtainStyledAttributes 获得一组值赋给 TypedArray（数组），这一组值来自于 res/values/attrs.xml 中的 name="CircleImageView"的 declare-styleable 中：

```
TypedArray a = context.obtainStyledAttributes(attrs,
        R.styleable.CircleImageView, defStyle, 0);
```

获取边界的宽度：

```
mBorderWidth = a.getDimensionPixelSize(
        R.styleable.CircleImageView_border_width, DEFAULT_BORDER_WIDTH);
```

获取边界的颜色

```
mBorderColor = a.getColor(R.styleable.CircleImageView_border_color,
        DEFAULT_BORDER_COLOR);
```

注意：这两种方法是将 ImageView 设置为圆形的关键点。

- 初始化：将图片水平垂直居中，进行缩放

```
super.setScaleType(SCALE_TYPE);
mReady = true;

if (mSetupPending) {
    setup();
    mSetupPending = false;
}
```

- 重写 onDraw 方法

绘制内圆形，图片、画笔为 mBitmapPaint

```
canvas.drawCircle(getWidth() / 2, getHeight() / 2, mDrawableRadius,
        mBitmapPaint);
```

如果圆形边缘的宽度不为 0 我们还要绘制带边界的外圆形 边界画笔为 mBorderPaint

```
if (mBorderWidth != 0) {
    canvas.drawCircle(getWidth() / 2, getHeight() / 2, mBorderRadius,
            mBorderPaint);
}
```

- setBorderColor 设置边框颜色

```
setBorderColor(getContext().getResources().getColor(borderColorRes));
```

- setBorderWidth 设置边框宽度

```
if (borderWidth == mBorderWidth) {
    return;
}
mBorderWidth = borderWidth;
setup();
```

- setup 函数：进行图片画笔边界画笔(Paint)一些重绘参数初始化，构建渲染器 BitmapShader 用 Bitmap 来填充绘制区域,设置样式以及内外圆半径计算等

```
private void setup() {
    // 因为 mReady 默认值为 false,所以第一次进这个函数的时候 if 语句为真进入括号体
内
    // 设置 mSetupPending 为 true 然后直接返回，后面的代码并没有执行。
    if (!mReady) {
        mSetupPending = true;
        return;
    }
    // 防止空指针异常
    if (mBitmap == null) {
        return;
    }
    // 构建渲染器，用 mBitmap 位图来填充绘制区域 ， 参数值代表如果图片太小的话 就
直接拉伸
    mBitmapShader = new BitmapShader(mBitmap, Shader.TileMode.CLAMP,
            Shader.TileMode.CLAMP);
```

```
    // 设置图片画笔反锯齿
    mBitmapPaint.setAntiAlias(true);
    // 设置图片画笔渲染器
    mBitmapPaint.setShader(mBitmapShader);
    // 设置边界画笔样式
    mBorderPaint.setStyle(Paint.Style.STROKE);// 设画笔为空心
    mBorderPaint.setAntiAlias(true);
    mBorderPaint.setColor(mBorderColor); // 画笔颜色
    mBorderPaint.setStrokeWidth(mBorderWidth);// 画笔边界宽度
    // 这个地方是取的原图片的宽高
    mBitmapHeight = mBitmap.getHeight();
    mBitmapWidth = mBitmap.getWidth();
    // 设置含边界显示区域，取的是 CircleImageView 的布局实际大小，为方形，查看 xml
也就是 160dp(240px)
    // getWidth 得到是某个 view 的实际尺寸
    mBorderRect.set(0, 0, getWidth(), getHeight());
    // 计算
    // 圆形带边界部分（外圆）的最小半径，取 mBorderRect 的宽高减去一个边缘大小的
一半的较小值（这个地方我比较纳闷为什么求外圆半径需要先减去一个边缘大小）
    mBorderRadius = Math.min((mBorderRect.height() - mBorderWidth) / 2,
            (mBorderRect.width() - mBorderWidth) / 2);
    // 初始图片显示区域为 mBorderRect（CircleImageView 的布局实际大小）
    mDrawableRect.set(mBorderRect);
    if (!mBorderOverlay) {
        // demo 里始终执行
        // 通过 inset 方法
        // 使得图片显示的区域从 mBorderRect 大小上下左右内移边界的宽度形成区域，
查看 xml 边界宽度为 2dp（3px），所以方形边长为就是 160-4=156dp(234px)
        mDrawableRect.inset(mBorderWidth, mBorderWidth);
    }
    // 这里计算的是内圆的最小半径，也即去除边界宽度的半径
    mDrawableRadius = Math.min(mDrawableRect.height() / 2,
            mDrawableRect.width() / 2);
    // 设置渲染器的变换矩阵也即是 mBitmap 用何种缩放形式填充
    updateShaderMatrix();
    // 手动触发 ondraw()函数 完成最终的绘制
    invalidate();
}
```

- ✧ 注意，该 CircleImageView 的使用方法和一般的 ImageView 一致，但是如果需要设置为圆形的 ImageView，必须在布局文件中设置 border_color 和 border_width（因为是由这两个参数去将该部件设置为圆形）

(5) 实现拍照或相册选取照片上传为三国人物头像

- ✧ 点击三国人物信息表的头像框，弹出 dialog，可选择是进行拍照还是选择相册图片：

```
new AlertDialog.Builder(this)
        .setTitle("设置头像...")
        .setNegativeButton("相册", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                Intent intent = new Intent(Intent.ACTION_PICK, null);
                intent.setDataAndType(
                        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
                        "image/*");
                startActivityForResult(intent, 1);
            }
        })
        .setPositiveButton("拍照", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                dialog.dismiss();
                Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                // 下面这句指定调用相机拍照后的照片存储的路径
                intent.putExtra(MediaStore.EXTRA_OUTPUT, Uri
                        .fromFile(imageFile));
                startActivityForResult(intent, 2);
            }
        }).show();
```

- 拍照：照片的路径、名称的设置：

为了防止拍照之后得到的相片的名称发生重复，采用根据手机系统时间（年月日时分秒）+“.jpg”来生成相片的名称。

路径的选取是根据当前环境的绝对路径+生成的文件名所得到。

```
Calendar now = new GregorianCalendar();
SimpleDateFormat simpleDate = new SimpleDateFormat("yyyyMMddHHmmss",
Locale.getDefault());
fileName = simpleDate.format(now.getTime())+".jpg";
filePath_takephoto =
Environment.getExternalStorageDirectory().getAbsolutePath()+ "/"+ fileName;
```

根据以上路径新建一个 ImageFile：

```
imageFile = new File(filePath_takephoto);
```

- 点击选择拍照后，dialog 消失:

调用相机进行拍照，并指定调用相机拍照后的照片存储的路径，跳转至 startActivityForResult()，注意，此处函数的第二个参数为 resultcode 而不是 requestCode。resultCode 判断是确定还是取消先,requestCode 再判断。RequestCode 和 ResultCode 的数值必须大于等于 0，同时在后面的设置中需记住相应的 RequestCode 和 ResultCode 以便信息的接收。

```
startActivityForResult(intent, 2);
```

- 在 dialog 中点击选择相册后:

dialog 消失:

使用这个方法可以调用，会让用户选择照片选取工具。系统会选取所有可用的

程序来供用户选择:

```
Intent intent = new Intent(Intent.ACTION_PICK, null);
```

设置数据和类型：

```
intent.setDataAndType(
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
        "image/*");
```

跳转至 startActivityForResult()函数中：

```
startActivityForResult(intent, 1);
```

- ✧ onActivityResult 接收所选图片信息：

  根据 Requestcode 进行判断是从相册中直接选择还是调用相机拍照：

  - ✧ 如果是直接从相册获取：

    android 多媒体数据库的封装接口:

    ```
    Cursor cursor = managedQuery(originalUri, proj, null, null, null);
    ```

    获得用户选择的图片的索引值:

    ```
    int column_index =
    cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA);
    ```

    将光标移至开头，防止引起越界：

    ```
    cursor.moveToFirst();
    ```

    最后根据索引值获取图片路径：

    ```
    filePath_photographers = cursor.getString(column_index);
    ```

  - ✧ 如果是调用相机拍照：

    根据所拍摄的图片的路径进行图像裁剪：

    ```
    File temp = new File(filePath_takephoto);
    startPhotoZoom(Uri.fromFile(temp));
    ```

  - ✧ 非空判断：

    要重新裁剪，丢弃：

    ```
    if (data != null) {
        setPicToView(data);
    }
    ```

- ✧ 裁剪图片方法：

  直接调用本地安卓库的图片裁剪功能：

  ```
  Intent intent = new Intent("com.android.camera.action.CROP");
  intent.setDataAndType(uri, "image/*");
  // 下面这个 crop=true 是设置在开启的 Intent 中设置显示的 VIEW 可裁剪
  intent.putExtra("crop", "true");
  ```

  设置比例：

  ```
  intent.putExtra("aspectX", 1);
  intent.putExtra("aspectY", 1);
  // outputX outputY 是裁剪图片宽高
  intent.putExtra("outputX", 150);
  intent.putExtra("outputY", 150);
  intent.putExtra("return-data", true);
  startActivityForResult(intent, 3);
  ```

- ✧ 保存裁剪之后的图片：

将 intent 中存储的图片信息转化为 bitmap:

```
Bitmap photo= extras.getParcelable("data");
```

将该 bitmap 型图片存储到手机内存指定位置中：

（该路径为上方预设的图片路径，若是选择拍照，拍照之后的图片会被裁剪后的图片所覆盖，但是选择相册图片，则不会覆盖原图）

```
try {
   File cut_file = new File(filePath_takephoto);
   FileOutputStream fos = new FileOutputStream(cut_file);
            photo.compress(Bitmap.CompressFormat.JPEG, 100, fos);
   fos.flush();
   fos.close();
} catch (FileNotFoundException e) {
   e.printStackTrace();
} catch (IOException e) {
   e.printStackTrace();
}
         cv.setImageBitmap(photo);
```

(6) 启动页 Viewpager 的实现

- ✧ 引导页面布局：整体布局由 RelativeLayout 实现，其中添加一个 ViewPage 部件，且 ViewPager 充满整个屏幕；

```
<android.support.v4.view.ViewPager
    android:id="@+id/guide_vp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</android.support.v4.view.ViewPager>
```

添加一个子 LinearLayout，用于 Viewpage 底部导航点的实现（该布局会在 java 文件中进行信息填充），并且设置为水平对中布局方式；

```
<LinearLayout
    android:id="@+id/guide_ll_point"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="40dp"
    android:layout_marginEnd="20dp"
    android:gravity="center_horizontal">
</LinearLayout>
```

在导航点的上方设置一个 ImageButton，初始化状态为隐藏，当检测到翻页翻到引导页面的最后一页时，ImageButton 显示。注意：当 ImageButton 控件 visibility 属性为 INVISIBLE 时，界面保留了 view 控件所占有的空间；而控件属性为 GONE 时，则界面不保留 view 控件所占有的空间。为防止在非最后一页的引导页误触该按钮，设置为：

```
android:visibility="gone"
```

- ✧ 重载继承 PagerAdapter 的 ViewPagerAdapter：

申请一个 View 型的 List 列表存储各界面。

getCount() 函数获取当前界面的数量：

```
@Override
public int getCount() {
    if (viewList != null){
        return viewList.size();
    }
    return 0;
}
```

isViewFromObject()判断对象是否生成界面：

```
@Override
public boolean isViewFromObject(View view, Object object) {
    return view == object;
}
```

根据输入数据的 index 初始化该位置的界面：

```
@Override
public Object instantiateItem(ViewGroup container, int position) {
    container.addView(viewList.get(position));
    return viewList.get(position);
}
```

destroyItem()删除界面：

```
@Override
public void destroyItem(ViewGroup container, int position, Object object) {
    container.removeView(viewList.get(position));
}
```

- ✧ 引导页面的 java 实现：
  - ✓ 加载 ViewPager：

    首先实例化图片资源，由于本应用的引导页数量设置为 3 页，存储图片资源的数组存储了 3 张 Viewpage 的背景图片的 R.mipmap.XXX。

    ```
    imageIdArray = new
    int[]{R.mipmap.welcome_1,R.mipmap.welcome_2,R.mipmap.welcome_3};
    ```

    获取一个 Layout 参数，设置为全屏:

    ```
    LinearLayout.LayoutParams params = new
    LinearLayout.LayoutParams(    LinearLayout.LayoutParams.MATCH_PARENT,Linear
    Layout.LayoutParams.MATCH_PARENT);
    ```

    根据上面设置的图片资源的数量设置相等的 View 集合：

    ```
    int len = imageIdArray.length;
    for (int i = 0;i<len;i++){
        //new ImageView 并设置全屏和图片资源
        ImageView imageView = new ImageView(this);
        imageView.setLayoutParams(params);
        imageView.setBackgroundResource(imageIdArray[i]);
        //将 ImageView 加入到集合中
    ```

```
    viewList.add(imageView);
}
```

初始化 ViewPage 的 Adapter:

```
vp.setAdapter(new ViewPagerAdapter(viewList));
```

设置滑动监听,绑定回调:

```
vp.setOnPageChangeListener(this);
```

- ✓ 加载导航点:

由于在布局文件中我们仅用了一个 LinearLayouts 实现导航点的大致声明，在此需要对布局信息进行初始化。导航点的数量有图片资源的数量保持一致，但是分为两类，当前页面的导航点为灰色圆点，其他页面的导航点为白色圆点。

```
ivPointArray = new ImageView[viewList.size()];
```

循环新建底部圆点 ImageView，将生成的 ImageView 保存到数组中。由于是导航点的初始化，此处将第一个页面的导航点设为灰色圆点，其他页面的导航点为白色圆点。

```
int size = viewList.size();
for (int i = 0;i<size;i++){
    iv_point = new ImageView(this);
    iv_point.setLayoutParams(new ViewGroup.LayoutParams(20,20));
    iv_point.setPadding(100,0,100,0);
    ivPointArray[i] = iv_point;
    //第一个页面需要设置为选中状态，这里采用两张不同的图片
    if (i == 0){
        iv_point.setBackgroundResource(R.drawable.point_1);
    }else{
        iv_point.setBackgroundResource(R.drawable.point_2);
    }
    //将数组中的 ImageView 加入到 ViewGroup
    vg.addView(ivPointArray[i]);
}
```

注意，导航点初始化之后需要添加进 view 集合中。

```
vg.addView(ivPointArray[i]);
```

- ✓ 当页面被滑动后对页面信息的更新:

获取当前页面的位置，并将该页面的导航点设为灰色圆点，其他页面的导航点为白色圆点。

```
int length = imageIdArray.length;
for (int i = 0;i<length;i++){
    ivPointArray[position].setBackgroundResource(R.drawable.point_1);
    if (position != i){
        ivPointArray[i].setBackgroundResource(R.drawable.point_2);
    }
}
```

判断该页面是不是最后一个页面，如果是则显示按钮

```
if (position == imageIdArray.length - 1){
    ib_start.setVisibility(View.VISIBLE);
}else {
    ib_start.setVisibility(View.GONE);
}
```

点击该按钮和正式进入本应用的 MainActivity 中：

```
ib_start = (ImageButton) findViewById(R.id.guide_ib_start);
ib_start.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new
Intent(MainActivity2.this,Update_Information.class);
        Bundle bundle = new Bundle();
        bundle.putString("begin","begin");
            //extras 根据改页面对应的人物信息进行修改
        intent.putExtras(bundle);
        startActivity(intent);
    }
});
```

(7) 数据库模块的理论基础：

采用 Android 提供的嵌入式关系数据库 SQLite，应用程序中的任何类都可以通过名称来访问该数据库。通过 SQLiteOpenHelper 实现对 SQLite 数据库的操作。

✧ SQLiteOpenHelper 的介绍：

SQLiteOpenHelper 是一个辅助类；

✓ 作用：管理数据库（创建、增、删、查、改）以及版本控制；

✓ 使用方法：通过创建子类继承 SQLiteOpenHelper，并实现它的一些方法来对数据库进行操作。

✓ 操作方法概览：

方法名	作用	备注
onCreate()	创建数据库	创建数据库自动调用（只有在安装一个软件并且调用 getReadableDatabase() 或 getWritableDatabase() 函数的时候，才会执行该函数，创建数据库）
onUpdate()	升级数据库	
close()	关闭所有打开的数据库对象	
execSQL()	执行给定的 SQL 语句	可进行增删改操作，不能进行查询操作
query()、rawQuery()	查询数据库	
insert()	插入数据	
delete()	删除数据	
getWritableDatabase()	创建或打开可以读/写的	以读写方式打开数据库，

	数据库	若数据库的磁盘空间满了，数据库就只能读而不能写，使用该方法打开数据库就会出错。
getReadableDatabase()	创建或打开可读的数据库	先以读写方式打开数据库，倘若使用如果数据库的磁盘空间满了，就会打开失败，当打开失败后会继续尝试以只读方式打开数据库.

✓ 方法详解：

- **onCreate**(SQLiteDatabase db)函数：创建数据库表
  将创建数据库表的 execSQL()方法和初始化表数据的一些 insert()方法写在该函数中。
  注：仅执行 db.execSQL(建表语句)，数据库实际上没有被创建或打开，直到 getWritableDatabase() 或者 getReadableDatabase() 方法中的一个被调用时才会进行创建或者打开，数据库文件位于/data/data/<包名>/databases。
- onUpgrade(SQLiteDatabse, int oldVersion, int newVersion)：更新数据库表结构
- 数据库版本发生变化时回调该函数（取决于数据库版本），如果 DATABASE_VERSION 的值被改为 2，系统发现现有数据库版本不同，就会调用该方法。例如：

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL(" ALTER TABLE person ADD phone VARCHAR(12) NULL "); //往表中增加一列
     // DROP TABLE IF EXISTS person 删除表
}
```

- **openOrCreateDatabase**(String path, SQLiteDatabase.CursorFactory factory): 根据给定条件连接数据库，如果此数据库不存在，则创建。
- **execSQL**(String sql): 执行给定 SQL 语句。

execSQL(String sql, Object[] bindArgs)方法的第一个参数为 SQL 语句，第二个参数为 SQL 语句中占位符参数的值，参数值在数组中的顺序要和占位符的位置对应。例：

```
SQLiteDatabase db = ....;
db.execSQL("insert into person(name, age) values(?,?)", new Object[]{"传智播客", 4});
db.close();
```

- **insert**(String table, String nullColumnHack, ContentValues values): 根据给定条件，插入一条记录。

```
SQLiteDatabase db = databaseHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("name", "林计钦");
values.put("age", 24);
long rowid = db.insert("person", null, values);//返回新添记录的行号，与主键id无关
```

不管第三个参数是否包含数据，执行 Insert()方法必然会添加一条记录，如果第三个参数为空，会添加一条除主键之外其他字段值为 Null 的记录。Insert()方法内部实际上通

过构造 insert SQL 语句完成数据的添加，Insert()方法的第二个参数用于指定空值字段的名称，如果第三个参数 values 为 Null 或者元素个数为 0， 由于 Insert()方法要求必须添加一条除了主键之外其它字段为 Null 值的记录，为了满足 SQL 语法的需要， insert 语句必须给定一个字段名，如：insert into person(name) values(NULL)，倘若不给定字段名 ，insert 语句就成了这样： insert into person() values()，显然这不满足标准 SQL 的语法。对于字段名，建议使用主键之外的字段，如果使用了 INTEGER 类型的主键字段，执行类似 insert into person(personid) values(NULL)的 insert 语句后，该主键字段值也不会为 NULL。如果第三个参数 values 不为 Null 并且元素的个数大于 0 ，可以把第二个参数设置为 null。

- **delete**()方法：删除满足要求的元组

```
SQLiteDatabase db = databaseHelper.getWritableDatabase();
db.delete("person", "personid<?", new String[]{"2"});
db.close();
```

- **query**(table, columns, selection, selectionArgs, groupBy, having, orderBy, limit)方法各参数的含义：

  table：表名。相当于 select 语句 from 关键字后面的部分。如果是多表联合查询，可以用逗号将两个表名分开。

  columns：要查询出来的列名。相当于 select 语句 select 关键字后面的部分。

  selection：查询条件子句，相当于 select 语句 where 关键字后面的部分，在条件子句允许使用占位符“?”

  selectionArgs：对应于 selection 语句中占位符的值，值在数组中的位置与占位符在语句中的位置必须一致，否则就会有异常。

  groupBy：相当于 select 语句 group by 关键字后面的部分

  having：相当于 select 语句 having 关键字后面的部分

  orderBy：相当于 select 语句 order by 关键字后面的部分，如：personid desc, age asc;

  limit：指定偏移量和获取的记录数，相当于 select 语句 limit 关键字后面的部分。

- **rawQuery**(String sql, String[] selectionArgs):根据给定 SQL，执行查询。

  第一个参数为 select 语句；第二个参数为 select 语句中占位符参数的值，如果 select 语句没有使用占位符，该参数可以设置为 null。带占位符参数的 select 语句使用例子如下：

```
Cursor cursor = db.rawQuery("select * from person where name like ? and age=?", new String[]{"%林计钦%", "4"});
```

```
SQLiteDatabase db = ....;
Cursor cursor = db.rawQuery("select * from person", null);
while (cursor.moveToNext()) {
   int personid = cursor.getInt(0); //获取第一列的值,第一列的索引从0开始
   String name = cursor.getString(1);//获取第二列的值
   int age = cursor.getInt(2);//获取第三列的值
}
cursor.close();
db.close();
```

Cursor 是结果集游标，用于对结果集进行随机访问。使用 moveToNext()方法可以将游标从当前行移动到下一行，如果已经移过了结果集的最后一行，返回结果为 false，否则为 true。另外 Cursor 还有常用的 moveToPrevious()方法（用于将游标从当前行移动到上一行，如果已经移过了结果集的第一行，返回值为 false，否则为 true）、moveToFirst()

方法（用于将游标移动到结果集的第一行，如果结果集为空，返回值为 false，否则为 true）和 moveToLast()方法（用于将游标移动到结果集的最后一行，如果结果集为空，返回值为 false，否则为 true）。

- **update**(String table, ContentValues values, String whereClause, String[] whereArgs): 根据给定条件，修改符合条件的记录。

```
SQLiteDatabase db = databaseHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("name", "林计钦");//key为字段名，value为值
db.update("person", values, "personid=?", new String[]{"1"});
db.close();
```

(8) 数据库的具体实现：

✧ 自定义一个继承 SQLiteOpenHelper 类的子类 DataBaseHelper，该类中由以下几部分组成：

✓ 私有变量，包括数据库名称、数据库版本、操作数据库的 SQLiteDatabase 类型的对象、数据表名称、初始化数据库所用到的数据：

```
private final static String DB_NAME = "figures.db";//数据库名字
private final static int VERSION = 1;//数据库版本
private SQLiteDatabase db;
private final static String TABLE_NAME = "information";

private int initial_size = 13;
private String[] photo = {"R.mipmap.liubei", "R.mipmap.simayi", "R.mipmap.xunyu", "R.mipmap.huatuo
        "R.mipmap.xiaoqiao", "R.mipmap.zhenmi", "R.mipmap.guanyu", "R.mipmap.zhangfei", "R.mipma
        "R.mipmap.zhugeliang", "R.mipmap.sunquan", "R.mipmap.zhouyu"};

private String[] name = {"刘备", "司马懿", "荀彧", "华佗", "曹操",
        "小乔", "甄宓", "关羽", "张飞", "赵云",
        "诸葛亮", "孙权", "周瑜"};
```

✓ 构造函数（有三个传参不同的构造函数），一个通过 super 调用父类的构造函数、一个传入参数包括上下文、数据库名称和数据库版本的构造函数、还有一个只传入上下文的构造函数：

```
// SQLiteOpenHelper子类必须要的一个构造函数
public DataBaseHelper(Context context, String name, CursorFactory factory,int version) {
    //必须通过super 调用父类的构造函数
    super(context, name, factory, version);
}
//数据库的构造函数，传递三个参数的
public DataBaseHelper(Context context, String name, int version){this(context, name, null, version)
//数据库的构造函数，传递一个参数的， 数据库名字和版本号都写死了
public DataBaseHelper(Context context){this(context, DB_NAME, null, VERSION);}
```

✓ onCreate()函数：

执行 sql 语句创建数据表，该表名称为 information，包含的属性有：

- _id integer：该表的键值：人物 id（自增型的）；
- photo String：图像在 Android 中的引用 id (R.mpimap.名称)或图像在手机内的存储地址；
- flag int：标志位，如果图像是在 android 的 mipmap 文件夹下，那么 flag=0；如果图片是在手机本地，那么 flag=1；
- name text：人物名字；
- sex text：人物性别；
- birth_to_death text：生卒年月；

- hometown text：人物籍贯；
- country text：效忠势力；
- introduction text：人物介绍；
- life int：人物生命值；
- force int：人物武力值。

```
// 执行sql语句完成数据库的创建
String CREATE_TBL = "create table information(_id integer primary key autoincrement, " +
                    "photo String, flag int, " +
                    "name text, sex text, birth_to_death text, hometown text, country text, introduction text, "+
                    "life int, force int);";
db.execSQL(CREATE_TBL);
// 数据库实际上是没有被创建或者打开的，直到getWritableDatabase() 或者 getReadableDatabase()
```

成功建表后，调用插入函数，将该数据表的初始信息插入进去。

```
// 插入一些初始化数据
// 创建ContentValues对象
for(int i=0; i<initial_size; i++)
{
    ContentValues values = getValues(photo[i], 0, name[i], sex[i], birth_to_death[i], hometown[i],
                                     country[i], introduction[i], life[i], force[i]);
    db.insert(TABLE_NAME, null, values);
}
```

✓ Insert()函数：

先打开数据库，然后执行 insert 语句，将 ContentValues 中需要插入的内容存入数据库；

```
//插入方法
public void insert(ContentValues values){
    //获取SQLiteDatabase实例
    db = getReadableDatabase();
    //插入数据库中
    db.insert(TABLE_NAME, null, values);
    // 第一个参数：要操作的表名称
    // 第二个参数：SQl不允许一个空列，如果ContentValues是空的，那么这一列被明确的指明为NULL值
    // 第三个参数：ContentValues对象
}
```

一系列查询函数，包括查询当前数据库中所有人物的名字、籍贯等信息；

通过人物名字查询人物信息等；

```
// 查询功能：通过名字查询表项
public List<Map<String,String>> GetItemFromName(String name){

    List<Map<String,String>> information = new ArrayList<>();

    db = getReadableDatabase();
    Cursor cursor = db.rawQuery("select _id, name, sex, country, photo, flag from information where

    while(cursor.moveToNext()){
        // 获取数据
        // 需要每个人物（元组）的头像、姓名、性别、生卒年月
        Map<String, String> tem = new LinkedHashMap<>();
        tem.put("Id",cursor.getString(0));
        tem.put("name",cursor.getString(1));
        tem.put("sex",cursor.getString(2));
        tem.put("country",cursor.getString(3));
        tem.put("photo",cursor.getString(4));
        tem.put("flag",cursor.getString(5));
        information.add(tem);
    }
    // 当用完时，要将cursor释放
    cursor.close();
    return information;
}
```

✓ Delete()函数：

先打开数据库，然后从数据库中删除对应 id 的人物；

```
//根据唯一标识_id ，来删除数据
public void delete(int id){
    db = getReadableDatabase();
    db.delete(TABLE_NAME, "_id=?", new String[]{String.valueOf(id)});
}
```

✓ update()函数：

更新数据库中的某些满足条件的元组属性值。

```
//更新数据库的内容
public void update(ContentValues values, int id){
    db = getReadableDatabase();
    // values里存放需要更改的列名及更改后的数值
    db.update(TABLE_NAME, values, "_id=?", new String[]{String.valueOf(id)});
}
```

✓ close()函数：

关闭数据库。

```
//关闭数据库
public void close(){
    if(db != null){
        db.close();
    }
}
```

✧ 数据库的应用：

在需要操作数据库的其他 activity 中：实例化 DatabaseHelper 得到变量 dbHelper，然后调用 getReadableDatabase()打开数据库，之后便可以用 dbHelper 来调用上述介绍的各个函数来达到操作数据库的目的，当操作都结束时，要执行 dbHelper.close()来关闭数据库。

```
// 创建数据库
DataBaseHelper dbHelper = new DataBaseHelper(this);
// 建立新表，调用getReadableDatabase()或getWritableDatabase()才算真正创建或打开数据库
SQLiteDatabase sqliteDatabase = dbHelper.getReadableDatabase();
```

(9) 人物 PK 模块：

✧ 点击“相框”内的“提示信息”加载英雄：

✓ 为“相框”内的“提示信息”设置监听器，一旦点击该信息，就会调用 startActivityForResult()函数从 PK 界面跳转到“英雄列表”界面：

注：由于有两个英雄需要被选择，所以，不同的英雄在调用 startActivityForResult()时，传入不同的请求码，方便得到英雄信息后的界面更新。

```
tips_1.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view)
    {
        Intent intent = new Intent(pkActivity.this, activity_pk_figures.class);
        startActivityForResult(intent, 0);
    }
});
tips_2.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view)
    {
        Intent intent = new Intent(pkActivity.this, activity_pk_figures.class);
        startActivityForResult(intent, 1);
    }
});
```

✓ 在“英雄列表”界面选择英雄：为 RecyclerView 的表项设置监听器，一旦点击选中的英雄，就会通过 setResult()函数将选中英雄的信息传回 PK activity，然后调用 finish()函数，跳转回 PK 界面：

```
commonAdapter.setOnItemClickListener(new CommonAdapter.OnItemClickListener() {
    @Override
    public void onClick(int position) {

        Map<String,String> tem = recyclerView_data.get(position);
        Intent intent = new Intent();
        intent.putExtra("photo",tem.get("photo"));
        intent.putExtra("flag",tem.get("flag"));
        intent.putExtra("name",tem.get("name"));
        intent.putExtra("life",tem.get("life"));
        intent.putExtra("force",tem.get("force"));
        setResult(RESULT_OK, intent);
        finish();
    }
    @Override
    public void onLongClick(int position) {
        // do nothing
    }
});
```

✓ 在 PK 的 activity 中重载 onActivytyResult()函数，来接收从英雄列表界面传来的选中英雄的信息，并根据得到的信息，对 PK 界面进行更新：

注：

- 由于“提示信息”的作用是：当我们进入 PK 界面，还没有选择英雄的时候给我们提示，所以一旦选好了英雄，便通过 setVisibility()把“提示信息”设置为不可见状态；
- 第一次选择英雄时，是通过点击“提示信息”跳转到英雄列表进行选择的，请求码设为 0；如果一局 PK 结束，想要换个英雄继续 PK，则需要点击“英雄头像“来跳转到英雄列表进行选择，请求码设为 2；所以当请求码为 0 或 2 时，得到的都是左边英雄的信息；
- 当一局 PK 结束时，英雄的血槽可能已经空了，如果此时点击“英雄头像“继续选择新的英雄，那么返回该界面时，英雄的血槽需要更新为满血状态，所以将 process1（掉血量）设为 0。

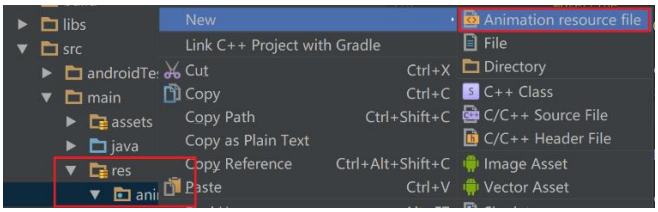
```java
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    Bundle extras = data.getExtras();
    String photo = extras.getString("photo");
    String flag = extras.getString("flag");
    String name = extras.getString("name");
    final String life = extras.getString("life");
    final String force = extras.getString("force");

    if(requestCode == 0 || requestCode == 2)
    {
        if(resultCode == RESULT_OK)
        {
            if(requestCode == 2){
                process1 = 0;
                handler.sendEmptyMessage(1);
            }
            force1 = Integer.parseInt(force);
            life1 = Integer.parseInt(life);
            // 将 tips_1 隐藏
            int vi = tips_1.getVisibility();
            if(vi == View.VISIBLE)vi = View.INVISIBLE;
            tips_1.setVisibility(vi);
```

✧ 人物平移的动画：

✓ 在 res 文件夹下建立“anim”文件夹，并在该文件夹下新建 Animation resource file：



✓ 左边英雄的平移动画属性如下：

注：总体的动画效果是：动画被触发 200ms 后，左边的英雄头像用 2s 时间移动距离自己原本位置右侧 600dp 处，然后立即原路返回自己原来的位置。

```xml
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"  设为匀速平移
    android:fillAfter="true">
    <translate
        android:repeatCount = "0"  动画不重复；
        android:startOffset = "200"  动画被触发后，等待200ms才开始；
        android:duration="2000"  动画持续时间：2s；
        android:fromXDelta="0"  动画开始时，物体相对于原本位置的距离：0dp;
        android:toXDelta="600"  动画结束时，物体相对于原本位置的距离：右侧600dp;
    />
    <translate
        android:repeatCount = "0"
        android:startOffset = "2200"  该动画的开始时间即是上一个动画的结束时间，也就是说，上一个动画刚结束，这个动画便开始了
        android:duration="2000"
        android:fromXDelta="0"
        android:toXDelta="-600"
        />
</set>
```

✓ 在 java 函数中的使用：先获得动画的主题 ImageView，然后通过上一步骤设置的动画属性产生动画效果，并将其作为参数传入到 AnimationUtils.loadAnimation 函数中：

```java
// 获得动画的变量
figure1 = (ImageView) findViewById(R.id.figure_1);
final ImageView figure_1bk = (ImageView) findViewById(R.id.figure_1bk);
final Animation translateAnimation = AnimationUtils.loadAnimation(this, R.anim.figure1_trasition);
figure2 = (ImageView) findViewById(R.id.figure_2);
final ImageView figure_2bk = (ImageView) findViewById(R.id.figure_2bk);
final Animation translateAnimation_2 = AnimationUtils.loadAnimation(this, R.anim.figure2_trasition);
```

当点击“PK 按钮”时，并且两位英雄的血槽都没有空的情况下，触发动画效果：

```java
// 设置监听器
ImageView pk = (ImageView) findViewById(R.id.text_button);
pk.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view) {
        // 判断血量
        if (process1 <= 10000 && process2 <= 10000) {
            // 点击图片
            // 设置动画，动画结束需要3s
            figure1.startAnimation(translateAnimation);
            figure_1bk.startAnimation(translateAnimation);
            figure2.startAnimation(translateAnimation_2);
            figure_2bk.startAnimation(translateAnimation_2);
```

- ✧ 气泡产生的动画：
  - ✓ 在 res->anim 文件夹下新建气泡的 Animation resource file，动画属性如下：
    总体动画效果：触发动画后，气泡慢慢隐现，然后不断上升，在上升过程中气泡越来越清晰，越来越大。

```xml
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"  所有动画效果都是匀速的
    android:fillAfter="true">
<translate
    android:repeatCount = "0"          平移效果：
    android:startOffset = "200"        触发动画后200ms开始向上平移；
    android:duration="5000"            5s的时间，向上平移1000dp
    android:fromYDelta="0"
    android:toYDelta="-1000"
/>
<alpha
    android:startOffset = "200"        透明度效果：
    android:fromAlpha="0.1"            触发动画后200ms透明度为0.1；
    android:toAlpha="1"                2s的时间，透明度变为1
    android:duration="2000"
/>
<scale
    android:startOffset = "200"
    android:fromXScale="0.5"           大小变化效果：
    android:fromYScale="0.5"           触发动画后200ms，气泡的半径为原本大小的一半；
    android:toXScale="2"               经过5s后，气泡的半径变为原本大小的2倍
    android:toYScale="2"
    android:duration="5000"
/>
</set>
```

  - ✓ 在 java 文件中的调用：
    Bubble 原本设置为“不可见”状态，只有在动画被触发时，才设为“可见”状态：

```
// bubble
final TextView bubble1 = (TextView) findViewById(R.id.bubble);
int bu_1 = bubble1.getVisibility();
if(bu_1 == View.VISIBLE)bu_1 = View.INVISIBLE;
bubble1.setVisibility(bu_1);
final TextView bubble2 = (TextView) findViewById(R.id.bubble2);
int bu_2 = bubble2.getVisibility();
if(bu_2 == View.VISIBLE)bu_2 = View.INVISIBLE;
bubble2.setVisibility(bu_2);
final Animation translateAnimation_3 = AnimationUtils.loadAnimation(this, R.anim.bubble_alpha);
```

根据人物血量的多少来设置气泡中的文本，即人物所说的话，然后触发气泡的动画：

```
// 血量减少
if(force1 < force2)
{
    if(process1 < 2000){
        bubble1.setText("发挥失常");
        bubble2.setText("承让了！");
    }
    else if(process1 < 4000) {
        bubble1.setText("怀疑人生");
        bubble2.setText("你个弱鸡");
    }
    else if(process1 < 6000) {
        bubble1.setText("又输了！！");
        bubble2.setText("又赢啦！");
    }
    else if(process1 < 8000) {
        bubble1.setText("命不久矣");
        bubble2.setText("吃嘛嘛香");
    }
    else {
        bubble1.setText("黄泉路上");
        bubble2.setText("本人最帅");
    }
    bubble1.startAnimation(translateAnimation_3);
    bubble2.startAnimation(translateAnimation_3);
```

当左边英雄的武力值 < 右边英雄的武力值时：

当左边英雄的掉血量 < 2000时：

左边英雄的气泡中显示：发挥失常

右边英雄的气泡显示：承让了！

- ✧ 血槽渐空的动画：
  - ✓ 在网上找了血量槽的图片：

  - ✓ 想要血槽变空，只能将血槽的底色图片覆盖在红色的血条上，从视觉上看，血槽是空的了：

  - ✓ 想要血槽呈现半空状态，则需要用到 clip，来将覆盖在血条上的底色图片进行剪切，便可以呈现血槽半空状态了：

- clip 实现：
- 在 res->drawable 中新建 Drawable Resource file，定义切割图片的属性：

```
<?xml version="1.0" encoding="utf-8"?>
<clip xmlns:android="http://schemas.android.com/apk/res/android"
    android:clipOrientation="horizontal"    剪切图片的方向：水平剪切
    android:drawable="@mipmap/blood_clip"   需要剪切的图片是blood_clip
    android:gravity="right">                从右侧开始剪切并显示
</clip>
```

在 java 文件中的使用：只需要通过 ClipDrawable 类型的变量调用 setLevel(int process)函数就可以控制血槽的血量了，其中 process 的取值在[0,10000]，process 越大，掉血量越多。

为了让血槽中的血量不在一瞬间变少，而是有过渡的缓慢减少，让我们可以肉眼观察到血量变化的过程，新开一个线程，每当 process 增加 50 时，通过 handler 更新主屏幕的血量视图，然后调用 thread.sleep()函数让线程停止 36ms，然后继续增加 process，重复上述步骤，直到 process 增加的总量达到了本次 PK 的掉血量，结束线程：

注：各个英雄在每次 PK 的掉血量是分别根据两个英雄各自的生命值和两个英雄的武力值之差计算得来的：掉血量 = 武力值之差/生命值 * 10000。

```
// 第一个人物掉血量为 force2-force1
final int lose = (int) ((force2 - force1)*1.0/(life1*1.0) * 10000);   掉血量计算

Thread thread = new Thread(new Runnable(){      新开线程
    @Override
    public void run(){
        int cnt = 0;
        for(int m=0; m<lose; m++) {
            cnt++;
            if(cnt == 50){
                cnt = 0;
                process1 += 50;                 掉血量每加50，就通过handler更新血量视图
                handler.sendEmptyMessage(1);
                try{
                    Thread.sleep(36);           让线程暂停36ms
                }catch (InterruptedException e){
                    e.printStackTrace();
                }
            }
        }
    }
});
thread.start();
```

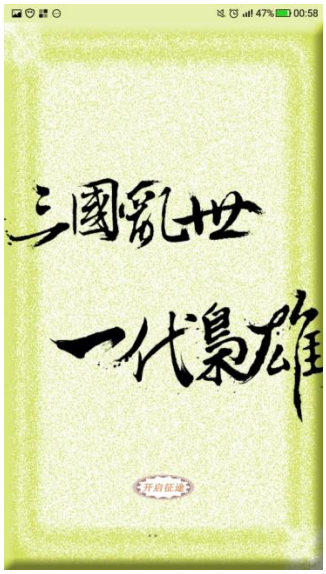
```
private Handler handler = new Handler(){
    @Override
    public void handleMessage(Message msg){
        switch(msg.what){
            case 1:
                clip1.setLevel(process1);//掉血process1
            case 2:
                clip2.setLevel(process2);//掉血process2
        }
    }
};
```

4、 代码架构


com.dictionaryofsanguo
Guide page
MainActivity2.java
ViewPagerAdapter.java
Service
MusicServer.java
UI
CircleImageView.java
CommonAdapter.java
MyViewHolder.java
DetailLayout.java
edit
MainActivity.java
Update_Information.java
search.java
database
DataBaseHelper.java
PK_funciton
pkActivity.java
activity_pk_figures.java

四、应用效果

1. 点击 app 图标进入应用，该应用有三个引导界面，如下图所示：



2. 点击“开启征途”按钮，进入应用主界面，主界面显示的主要是三国英雄人物列表，如下图所示：

3. 长按人物列表时，会弹出一个对话框，让你选择是否删除该英雄人物，如下图所示：

4. 点击标题栏的搜索按钮，可以进入搜索界面，如下图是搜索“周瑜”出来的效果图，

5. 点击该人物可以进入详细信息界面，如下图所示：

6. 点击编辑按钮，进入修改界面，在这个界面可以实现在原有的基础上对英雄人物信息的修改，该界面如下图所示：

7. 同时也可以自己添加英雄人物，点击主界面的“+”按钮，进入人物增加界面，如下图所示，人物头像的选择有两种方式，一种是拍照，另一种是相册。

8. 点击应用主界面工具栏的“Vs”进入人物 PK 界面
   - 进入 PK 界面：

- 点击“相框”，会跳转到英雄列表界面，选择想要进行 PK 的英雄：

- 选择之后，跳转回 PK 界面，此时选中的英雄信息也会被加载到 PK 界面，包括英雄名字、生命值、武力值以及英雄照片：

- PK 英雄加载成功后，可以点击“杀”按钮，那么两个英雄会进行一次 PK，PK 的过程是动态的，此时界面会有三个动态过程：1. 两个英雄图片同时向中间移动，相撞后再返回原位，代表着激烈的厮杀；2. 武力值弱的英雄会以肉眼可见的缓慢速度逐渐掉血；3.两个英雄附近会产生绿色气泡，气泡逐渐上升的同时也慢慢变大，并且

由透明状态渐渐清晰，气泡中是人物的互动交谈。

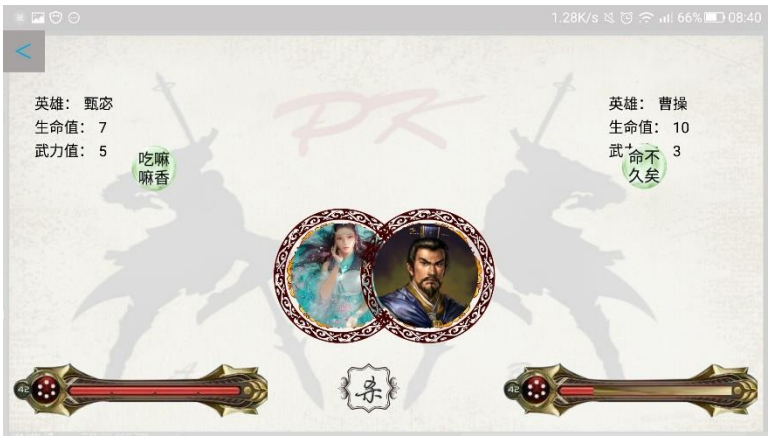
✓ 武力值不同的英雄的厮杀过程：

第一次 PK，可以看到武力值为 3 的曹操与武力值为 5 的甄宓 PK 的结果是曹操战败，血条减少；第一次输，曹操的反应是自己“发挥失常”，而甄宓也比较谦虚，并没有因此嘲笑曹操，只说了句“承让了！”；这是在 PK 前期截的屏，两个英雄头像正在向中间移动的过程中，绿色气泡还比较小，比较透明：

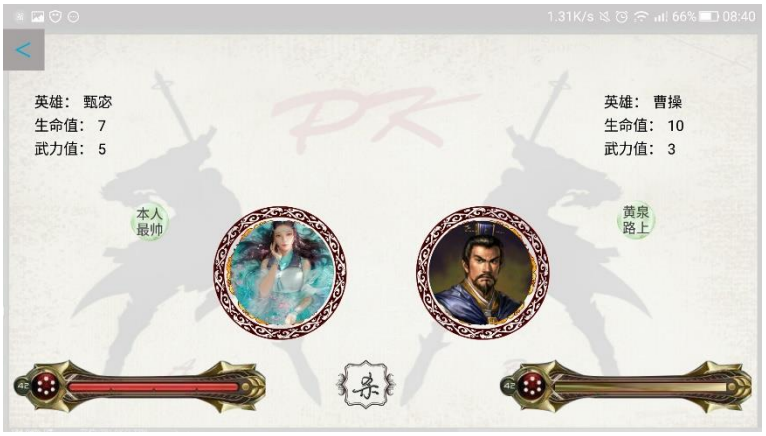
第二次 PK，曹操依旧掉血，所以他开始“怀疑人生”，而甄宓也忍不住嘲讽了起来，说道“你个弱鸡”；这是在 PK 中期截的屏，可以看到两个英雄的头像已经撞击在一起了，并且绿色气泡相对于上次的，更大更清晰了：

第三次 PK，毫无悬念，输的依旧是曹操，但根据他的反应来看，很明显，他已经接受了自己的败局，只剩下些许哀怨：

第四次 PK，曹操的血槽已经快空了，他似乎预感到了自己将要死去的悲惨结局：



第五次 PK，曹操的血槽彻底空了，他已经在黄泉路上了：

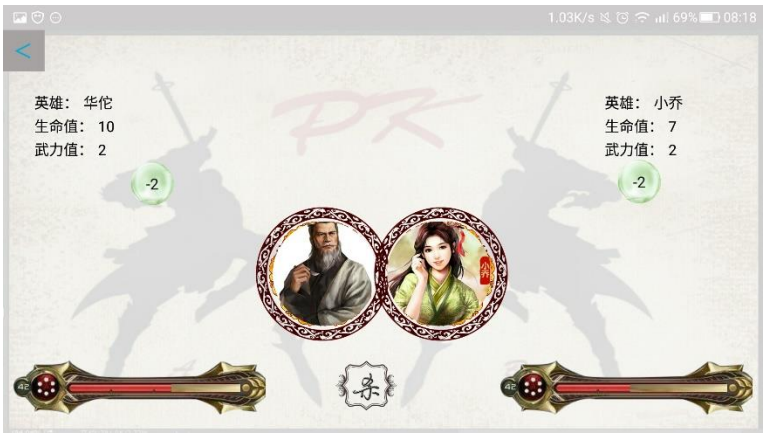


如果，你还想继续 PK，系统会提示你，已有人死亡，PK 无法继续：

但如果你觉得还没玩够，还可以点击英雄头像，继续选择英雄进行下一轮 PK。

✓ 武力值相同的英雄的厮杀过程：

由于华佗和小乔的武力值相当，所以在 PK 时，是两败俱伤的，两个人都会掉血：



虽然两个英雄的武力值相当，但华佗的生命值比小乔高，所以一直 PK 下去的话，最终的赢家会是华佗：

此时小乔血槽已空，华佗胜出：

五、开发过程中遇到的问题及解决办法

1、刚测试数据库功能时，无论如何也没办法创建数据库，明明我查到的资料上说的是：第一次创建数据库时会调用 onCreate()方法，但我写在 onCreate()函数中的建表语句并没有起到作用，于是便用 Log.d 输出信息来测试，结果发现程序根本就没有运行到 onCreate()函数，纠结了很久后，询问了同学才知道，原来只有在安装软件时，才会执行 SQLiteOpenHelper 类的 onCreate()函数，而我只是重启了软件，当然不会执行该函数，于是我将手机中的 app 卸载，又运行了一次程序，数据库就被成功创建了！

2、我做完 PK 界面后，再运行程序就一直报错：“Error:Execution failed for task ':app:mergeDebugResources'. > Crunching Cruncher scrollbar_thumb.9.png failed, see logs”，然后百度，再 app.gradle 中添加了以下代码：

```
aaptOptions {
    cruncherEnabled = false
    useNewCruncher = false
}
```

问题就解决了。

3、设置 PK 界面的气泡中显示信息时，有一种情况是要显示数字的，于是，我直接调用了 setText 函数，并传入了 int 类型的参数，然后运行就报错，显示的错误信息是：没有相应资源。复制了报错信息百度了一下，才知道原因：setText 函数的参数有两种：一种是字符串，另一种是整型；如果传入参数是字符串，则直接赋值到 setText 中，如果传入参数是整型，则会去 resource 中根据整型查找对应的字符串，然后转化为字符串。而我直接传入 int 类型的参数，虽然编译时不会报错，但运行时，TextView 的 setText 方法把传入的 int 类型的 percent 当做资源 Id 到项目中查询资源，而资源中却找不到相应的数值，就会报 NotFoundException 的错误。于是我将 int 类型的值用 String.valueOf（）函数转化为字符串，然后再作为参数传入 setText 函数，便不会报错了。

4、在实现从相册中选择照片上传为头像的这项功能时，我卡在了如何获得相册所选择的相片的具体路径这一问题中。不同于拍照上传头像这一功能可以事先根据当前环境的绝对路径加上当前系统时间的文件名获得照片的具体路径，相册选择图片我只

能定位到相册的具体路径，但是无法定位到具体是哪张图片哪个文件名。后来经过百度搜索并结合自身编写的代码，我使用了 Cursor 来获得用户选择的图片的索引值：

```
column_index = cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA);
filePath_photographers = cursor.getString(column_index);
```

但是添加了上面的代码之后，在我自己的手机上是能够正常运行的，但是在队友的手机中却会报错。为什么？

解决办法：添加光标获取图片的索引值之后要将光标移至开头 ，这个很重要，不然一不小心很容易引起越界。

5、 在完成了拍照、相册选取图片之后，我将新头像的路径保存在数据库中，但是问题又来了，路径是正确的，数据库调用方法也没有出错，但是就是无法将图片进行显示，这是为什么？

一开始我以为是自己将路径转换为图像发生了错误，采用了三种方法：1、根据路径得到 Uri；2、根据路径转化为 Bitmap；3、借用第三方架构 Glide；但是使用了以上的几种方法之后，还是无法显示图片。

后来，经同学的提醒，既然不是图片或数据库的问题，就很有可能是应用没有获取相应的权限来读取图片并显示。因此我对动态、静态权限的获取进行了检查。发现，是静态获取权限的放置位置发生了错误。我一开始惯性地将静态权限声明放在了 Manifest 的 application 里面，但是事实上必须放到 application 之后。经过以上的方法，我们终于能够正确地显示拍照、相册选取了的图片了。

六、思考及感想

1、 本次安卓应用开发的 project 给我最大的感悟就是团队之间的合作和沟通。一开始的时候我们小组对大致的工作内容进行了简单的分工，但是此时并没有真正地意识到分工的真正含义——各做各的功能项，但是却并没有很好地处理各个 Activity 之间的跳转关系，函数没有很好地封装，不方便队友的调用……种种问题，导致我们在实现了本应用软件的各项功能之后又花费了一定的时间去进行代码之间的整合、兼容。

因此，在实际的应用开发环境中，一定要做好模块化的工作，提高代码的可移植性。实际上，队友不会花费太多的时间来理解你编写的模块的代码，我们应该做到的是，明确编写的函数的输入和输出，不需要队友理解自己的代码，但是必须准确明了地表达出该函数是怎么被调用的，调用了之后会达到什么样的效果。

其次，我认为我们小组一开始对本期中项目的定位不够深刻。一开始只是想着：啊，要完成项目要求的某具体功能，目标是完成一款满足要求的 app；而不是真正地把这款 app 当成一种具有广泛流传可能性的应用软件。目标的不同，导致我们一开始完成 project 的基本功能时显得特别稚嫩：包括 UI 界面设计、功能设计等等。之后，看到有同学在朋友圈中晒出各自的 app 时，我们才意识到差距，意识到不足，才开始进一步地优化。

同样，在本次期中项目的 project 中，我还深深感觉到了美工的重要性。一开始以为在网上随便找几张图片作为背景就行了，但是实际操作中明显意识到界面的不美观。最后我只能硬着头皮 5 分钟 PS 入门速成，并逐渐在后来的界面的优化工作中对 PS 产生了浓厚的兴趣。

最后总结以下，这次 project 很有趣，特别是成品出来的那一刻，意义非凡。

——郑海君

2、

这是我们第一次以小组的形式做一个 project，所以有很多地方都处理得不是很有经验，比如，我们各自电脑上的包名都不同，

所以每次代码同步的时候，都要重新修改包名，虽然是很简单的事，但总是觉得多了些不必要的操作。

本次 project 相比于之前的作业还是比较难入手的，因为没有代码提示，没有具体细致的要求，一切都要自己构思，自己想效果，

然后再想办法实现，不过好在很多东西都可以借鉴之前的作业资料。其实对于 SQLite 数据库，现在我是理解了，便觉得很容易，但

刚开始了解这个知识点的时候，并不是那么容易就能接受的。我们原本想的是，要自己用 SQLite Studio 产生初始数据库的.db 文件，

然后将该文件放在 res->raw 文件夹下，在代码中使用文件写入操作将.db 文件写入手机的 data/包名目录下，然后再读取，

使用 SQLiteDatabase 类提供的方法对该数据库进行操作。我最开始是这样想的，也在自己的电脑上实现了，可以正确的对导入的数据库进行

增删查改操作，但将这一部分的代码移植到队友的 app 中，便总会有错误，找 bug 找了挺久的，还是没找出来，最后不得已，只好又写了

一个继承 SQLiteOpenHelper 类的 DatabaseHelper 子类，直接在代码中生成数据库，建立数据表，就不涉及文件操作了，最后才能在队友

的 app 中正常运行。所以在数据库这一方面，我们耗费了大量的时间。

其实在数据库做好之后，剩下的工作，进展的还算顺利，我还负责了 PK 功能的实现，从 PK 规则的构思到动画效果的表现，这个过程中，我查了

很多资料，也接触了一些效果的实现方法，其中用 clip 来实现血槽中血量变化的方法，我觉得很神奇。

不过最难的还是 UI 设计，虽然设计一个 UI 是容易的，但如果想让它很好看，就没那么简单了。要考虑背景、各个部件的协调，有时在网上找的

图标很好看，但不是透明的，还要自己抠图，这个过程让我深深地体会到了 ps 的重要性。

总之，虽然我们做的 app 在大家看来可能很 low，但自己一步一步做下来，成就感还是有的，这次 project 也让我对 android 开发兴趣大增，觉得这真的是

一件很有意思的事情！

——赵钰莹

3、

因为这次是第一次小组一起完成一个 project，所以代码在综合的时候总会出现各种问题，因为我主要是负责把所有的功能整合起来，所以在实验的过程中，就需要不断地调整代码来使其适应整个 project。同时，在完成这个 project 的过程中，我深深地觉得界面的设计不是一件简单的事，所以在设计界面之前，小组间的讨论很重要。

整个应用所需要添加的事件其实还挺多的，所以我在实现这些事件的时候，总会有一种很混乱的感觉，直到我画出了整个实验流程以及它们之间的信息传递关系，思路才逐渐清晰起来。

因为这次实验所涉及的很多内容都是我们从未涉及过的，所以，在完成 project 的时候，就需要查找很多的资料，所以，在完成实验之后，感觉自己学习到了很多知识，虽然我们的 project 效果可能不是那么好，但是这也是我们小组所有成员一起努力的成

果，所以看到我们的 app 能够正常地实现我们想要的功能，我还是很高兴的。
在整个实验过程中，虽然经常出现因为某些特别琐碎的错误而导致程序崩溃的情况，但是经过小组间一起 debug 从而修正错误，也是一件很有成就感的事。
——郑晓如

七、小组分工

学号	姓名	实验分工
15352433	赵钰莹	1、数据库功能的实现：包括数据库的创建、增删查改功能的实现； 2、PK 功能的实现； 3、界面优化：界面素材的搜索。
15352437	郑海君	1、 引导页面的设计与实现； 2、 三国人物详细信息页面的界面设计和功能实现； 3、 拍照或相册选取图片上传头像功能的实现； 4、 界面优化:自定义圆形头像、PS 技术担当。
15352440	郑晓如	1、 人物列表主界面的设计与实现； 2、 音乐播放功能的实现； 3、 整合代码、完成个 Activity 之间的跳转，加强组内代码的兼容性； 4、 界面优化：搜索界面素材，重调部件布局。

八、参考资料

1、 SQLite 数据库增删改查操作：
http://www.cnblogs.com/linjiqin/archive/2011/05/26/2059182.html
2、 Android 动画(Animation)：
http://blog.csdn.net/cq0072008/article/details/8223878
3、 Android：SQLiteOpenHelper 类（SQLlite 数据库操作）详细解析：
http://blog.csdn.net/carson_ho/article/details/53241633
4、 Android 数据的四种存储方式：
http://www.cnblogs.com/smalltigerlee/archive/2011/11/10/2244143.html
5、 利用 Service 实现背景音乐的播放：
http://blog.csdn.net/chenjie19891104/article/details/6330720
6、 Android 中保存图片到本地功能实现
http://blog.csdn.net/Samuel__Liu/article/details/50743106
7、 自定义圆形头像 CircleImageView 的使用和源码分析:
http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2015/0806/3268.html
8、 Android 实现自定义引导页，玩转 ViewPager: Android 开发：
http://www.jianshu.com/p/adb21180862a

9、 相册读取、拍照、图片裁剪和图片上传服务器等功能的实现:
http://blog.csdn.net/chentravelling/article/details/51292137

九、报告其他要求

期中项目的应用需以录制视频方式对实际操作和效果进行讲解。