1.Android 与服务器进行交互

Http 请求头 http://tools.jb51.net/table/http_header

```
public static String getSessionID(String userName, String password) {
    OutputStream os;
    HttpURLConnection con;
    String sessionID = "";
    URL realUrl;
    try {
        realUrl = new URL("http://seam.ustb.edu.cn:8080/jwgl/Login");
        con = (HttpURLConnection) realUrl.openConnection();
        // 设置POST方法
        con.setRequestMethod("POST");
        // 设置请求头
        con.setRequestProperty("Accept", "text/html");
        con.setRequestProperty("Connection", "Keep-Alive");
        con.setInstanceFollowRedirects(false); // 禁止响应3**类型重定向
        // 设置允许输入输出流
        con.setDoInput(true);
        con.setDoOutput(true);
        String str = "username=" + userName + "&password=" + password
                + "&usertype=student";
        con.connect();
        os = con.getOutputStream();
        // 把提交数据以输出流的方式写到服务器
        os.write(str.getBytes());
        os.flush();
        os.close();
        con.disconnect();
        sessionID = con.getHeaderField("Set-Cookie");
        sessionID = sessionID.substring(0, sessionID.indexOf(";"));
    } catch (Exception e) {
        e.printStackTrace();
    }
    return sessionID;
}
```

一定要设成false，否则自动重定向，session失效导致后来无法请求数据

获取到sessionID，需注意进行处理

```java
public static void main(String[] args) {
    try {
        OutputStream os;
        HttpURLConnection con;
        InputStream is;
        String sessionID = getSessionID("41356021", "zhRU19950804@");
        URL realUrl;
        realUrl = new URL("http://seam.ustb.edu.cn:8080/jwgl/index.jsp");
        con = (HttpURLConnection) realUrl.openConnection();
        con.setRequestProperty("Cookie", sessionID);
        con.setRequestProperty("Accept", "text/html");
        con.setDoInput(true);
        con.setDoOutput(true);
        con.connect();
        // 通过返回码判断是否连接成功
        if (con.getResponseCode() == 200) {
            // 获得服务器返回字节流
            is = con.getInputStream();
            // 创建本地文件
            os = new FileOutputStream("D:/a.txt");
            byte data[] = new byte[1024];
            int len;
            while ((len = is.read(data)) != -1) {
                os.write(data, 0, len);
            }
            os.flush();
            os.close();
            is.close();
            con.disconnect();
            System.out.println("fwdfasd");
        }
    } catch (Exception e) {

    }
}
```

添加请求头，保持会话状态

2. AsyncTask

AsyncTask 定义了三种泛型类型 Params，Progress 和 Result。

•Params 启动任务执行的输入参数，比如 HTTP 请求的 URL。

•Progress 后台任务执行的百分比。

•Result 后台执行任务最终返回的结果，比如 String。

doInBackground(Params...) 后台执行，比较耗时的操作都可以放在这里。注意这里不能直接操作 UI。此方法在后台线程执行，完成任务的主要工作，通常需要较长的时间。在执行过程中可以调用 publicProgress(Progress...)来更新任务的进度。

•onPostExecute(Result) 相当于 Handler 处理 UI 的方式，在这里面

可以使用在 doInBackground 得到的结果处理操作 UI。 此方法在主线程执行，任务执行的结果作为此方法的参数返回

onProgressUpdate(Progress…) 可以使用进度条增加用户体验度。此方法在主线程执行，用于显示任务执行的进度。

•onPreExecute() 这里是最终用户调用 Excute 时的接口，当任务执行之前开始调用此方法，可以在这里显示进度对话框。

•onCancelled() 用户调用取消时，要做的操作

使用 AsyncTask 类，以下是几条必须遵守的准则：

•Task 的实例必须在 UI thread 中创建；

•execute 方法必须在 UI thread 中调用；

• 不要手动的调用 onPreExecute(), onPostExecute(Result)，doInBackground(Params...), onProgressUpdate(Progress...)这几个方法；

•该 task 只能被执行一次，否则多次调用时将会出现异常；

使用方法，ui 线程中创建此对象，调用 execute()方法

3. Fragment（未完，附带回调函数）

http://blog.csdn.net/harvic880925/article/details/44917955

3.1 优点

Fragment 可以使你能够将 activity 分离成多个可重用的组件，每个都有它自己的生命周期和 UI。

•Fragment 可以轻松得创建动态灵活的 UI 设计，可以适应于不同的屏幕尺寸。从手机到平板电脑。

•Fragment 是一个独立的模块,紧紧地与 activity 绑定在一起。可以运行中动态地移除、加入、交换等。

•Fragment 提供一个新的方式让你在不同的安卓设备上统一你的 UI。

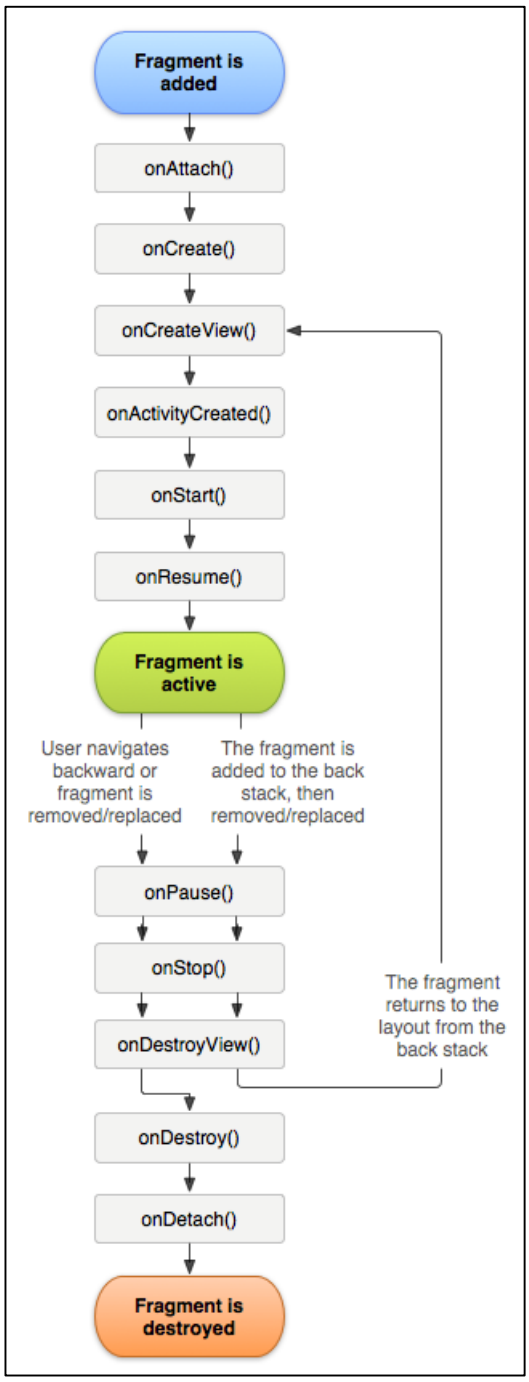
•Fragment 解决 Activity 间的切换不流畅，轻量切换。

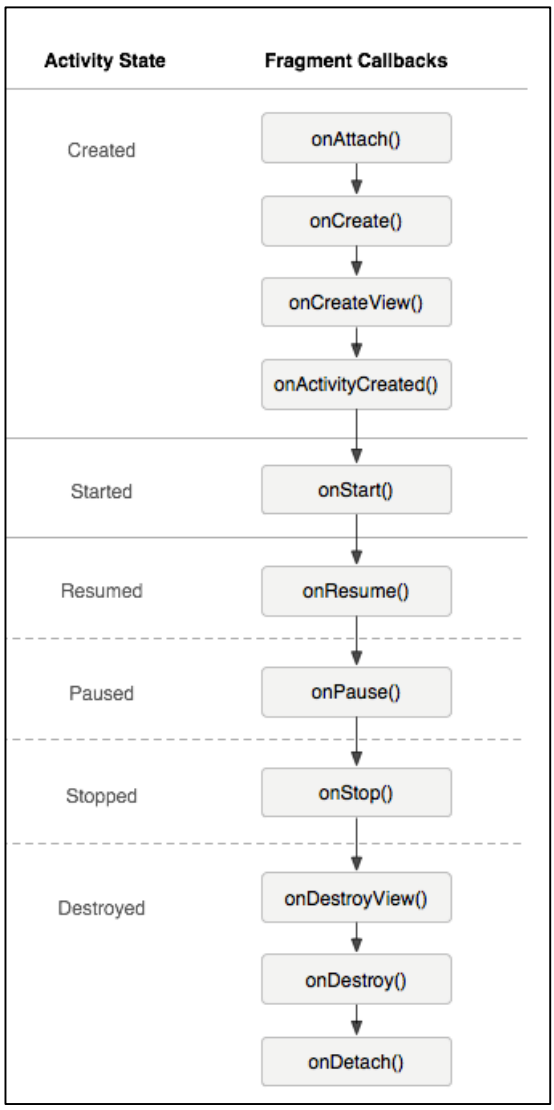
•Fragment 替代 TabActivity 做导航，性能更好。

•Fragment 在 4.2.版本中新增嵌套 fragmeng 使用方法，能够生成更好的界面效果。

•Fragment 做局部内容更新更方便，原来为了到达这一点要把多个布局放到一个 activity 里面，现在可以用多 Fragment 来代替，只有在需要的时候才加载 Fragment，提高性能

3.2 生命周期


Fragment is added
onAttach()
onCreate()
onCreateView()
onActivityCreated()
onStart()
onResume()
Fragment is active
User navigates backward or fragment is removed/replaced
The fragment is added to the back stack, then removed/replaced
onPause()
onStop()
onDestroyView()
The fragment returns to the layout from the back stack
onDestroy()
onDetach()
Fragment is destroyed



3.3 静态添加 Frament

使用`android.support.v4.app.FragmentActivity;`中 Fragment 功能更强大

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_blue_bright"
    >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is fragment 1"
        android:id="@+id/textView"
        android:layout_gravity="center_horizontal" />
</LinearLayout>
```

新建一个 fragment 布局文件

```java
/**
 * Created by jstxzhangrui on 2016/9/18.
 */
public class Fragment1 extends Fragment {
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment1, container, false);
    }
}
```

继承 Fragment，新建 MyFragment 类

```xml
    <fragment
        android:id="@+id/fragment1"
        android:name="com.ustb.zhangrui.fragment.Fragment1"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="1"
        />
```

在 activity 布局中添加 fragment，注意这里一定要给 fragment 添加 Id，否则加载不成功

3.4 动态加载 Fragment

1. 获取到 FragmentManager，在 V4 包中通过 getSupportFragmentManager，在系统中原生的 Fragment 是通过

getFragmentManager 获得的。

•2.开启一个事务，通过调用 beginTransaction 方法开启。

•3.向容器内加入 Fragment，一般使用 add 或者 replace 方法实现，需要传入容器的 id 和 Fragment 的实例。

•4.提交事务，调用 commit 方法提交。

注意使用 FrameLayout 来作为装载 Fragment 的容器

```
android:id="@+id/btn2" />
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/fra_container"
    >
</FrameLayout>
```

```
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.btn1:
            FragmentManager  fm = getSupportFragmentManager();
            FragmentTransaction transaction =fm.beginTransaction();
            Fragment1 f1 = new Fragment1();
            transaction.replace(R.id.fra_container,f1);
            transaction.commit();
            break;
        case R.id.btn2:
            FragmentManager  fm2 = getSupportFragmentManager();
            FragmentTransaction transaction2 =fm2.beginTransaction();
            Fragment2 f2 = new Fragment2();
            transaction2.replace(R.id.fra_container,f2);
            transaction2.commit();
            break;
    }
}
```

3.5 Fragment 的管理

1. FragmentManager

要管理 activity 中的 fragments，你就需要使用 FragmentManager。

通过 getFragmentManager（）或 getSupportFragmentManager（）获得

常用的方法有

manager.findFragmentById(); //根据 ID 来找到对应的 Fragment 实例，主要用在静态添加 fragment 的布局中，因为静态添加的 fragment 才会有 ID

manager.findFragmentByTag();//根据 TAG 找到对应的 Fragment 实例，主要用于在动态添加的 fragment 中，根据 TAG 来找到 fragment 实例

manager.getFragments();// 获取所有被 ADD 进 Activity 中的 Fragment

2. FragmentTransaction

一般用来对当前的 Fragment 进行管理，包括 add,replace,remove

//将一个 fragment 实例添加到 Activity 的最上层

add(int containerViewId, Fragment fragment, String tag);

//将一个 fragment 实例从 Activity 的 fragment 队列中删除

remove(Fragment fragment);

// 替换 containerViewId 中的 fragment 实例，注意，它首先把 containerViewId 中所有 fragment 删除，然后再 add 进去当前的 fragment

replace(int containerViewId, Fragment fragment);

3. 事务回滚处理

要使用回滚功能，只需要要使用下面两个代码：

在transaction.commit()之前，使用addToBackStack()将其添加到回退栈中。

```java
transaction.addToBackStack(String tag);
```

在需要回退时，使用popBackStack()将最上层的操作弹出回退栈。

```java
manager.popBackStack();
```

这里的popBackStack()是弹出默认的最上层的栈顶内容。

当栈中有多层时，我们可以根据id或TAG标识来指定弹出到的操作所在层。函数如下：

```java
void popBackStack(int id, int flags);
void popBackStack(String name, int flags);
```

其中

- 参数int id是当提交变更时transaction.commit()的返回值。
- 参数string name是transaction.addToBackStack(String tag)中的tag值；
- 至于int flags有两个取值：0或FragmentManager.POP_BACK_STACK_INCLUSIVE；
- 当取值0时，表示除了参数一指定这一层之上的所有层都退出栈，指定的这一层为栈顶层；
- 当取值POP_BACK_STACK_INCLUSIVE时，表示连着参数一指定的这一层一起退出栈；

2、回退栈（back stack）状态改变监听

FragmentManager还为我们提供了监控回退栈状态改变的方法：

```java
FragmentManager::addOnBackStackChangedListener(listener);//添加监听器
FragmentManager::removeOnBackStackChangedListener(listener);//移除监听器
```

通过添加监听器，就可以在回退栈内容改变时，及时收到通知；

3. 处理函数

一、hide()、show()

1、基本使用

这两个函数的功能非常简单，

```
[java]
01. public FragmentTransaction hide(Fragment fragment);//将指定的fragment隐藏不显示
02. public FragmentTransaction show(Fragment fragment);//将以前hide()过的fragment显示出来
```

3.6 Fragment 与 Activity 通信

a、如果你 Activity 中包含自己管理的 Fragment 的引用，可以通过引用直接访问所有的 Fragment 的 public 方法

b、如果 Activity 中未保存任何 Fragment 的引用，那么没关系，每个 Fragment 都有一个唯一的 TAG 或者 ID，可以通过 getFragmentManager.findFragmentByTag()或者 findFragmentById()获得任何 Fragment 实例，然后进行操作。

c、在 Fragment 中可以通过 getActivity 得到当前绑定的 Activity 的实例，然后进行操作。

注：如果在 Fragment 中需要 Context，可以通过调用 getActivity(),如果该 Context 需要在 Activity 被销毁后还存在，则使用 getActivity().getApplicationContext()。

考虑 Fragment 的复用，必须降低 Fragment 与 Activity 的耦合，而且 Fragment 更不应该直接操作别的 Fragement.

安卓进阶（案例）——菜鸟到高手

4. ListView 滑动删除案例

1.1 知识点技术

1. popupWindow

PopupWindow 这个类用来实现一个弹出框，可以使用任意布局的 View 作为其内容，这个弹出框是悬浮在当前 activity 之上的。

PopupWindow 与 AlertDialog 的区别

最关键的区别是 AlertDialog 不能指定显示位置，只能默认显示在屏幕最中间（当然也可以通过设置 WindowManager 参数来改变位置）。而 PopupWindow 是可以指定显示位置的，随便哪个位置都可以，更加灵活。

（1）构造函数

PopupWindow 没有默认布局，它不会像 AlertDialog 那样只 setTitle, 就能弹出来一个框。PopupWindow 是没有默认布局的，它的布局只有通过我们自己设置才行。

Pop 构造函数有四个，但要生成一个 PopupWindow 最基本的三个条件是一定要设置的：View contentView，int width, int height ；少任意一个就不可能弹出来 PopupWindow

PopupWindow 没有默认布局，它不会像 AlertDialog 那样只 setTitle, 就能弹出来一个框。PopupWindow 是没有默认布局的，它的布局只有通过我们自己设置才行，所以必须要有 View

(2)显示函数

popupWindow 宽和高以代码为准

（3）常用函数

dimiss()

用于不需要时，将窗体隐藏掉

setTouchable(boolean touchable)

设置 PopupWindow 是否响应 touch 事件，默认是 true

setFocusable(boolean focusable)

该函数的意义表示，PopupWindow 是否具有获取焦点的能力，默认为 False。一般来讲是没有用的，因为普通的控件是不需要获取焦点的，而对于 EditText 则不同，如果不能获取焦点，那么 EditText 将是无法编辑的。

setOutsideTouchable(boolean touchable)

PopupWindow 以外的区域是否可点击，即如果点击 PopupWindow 以外的区域，PopupWindow 是否会消失。必须配合 setBackgroundDrawable(Drawable background)使用

从代码来看没并没有真正设置 Bitmap，而只是 new 了一个空的 bitmap，原因在下面介绍。

setBackgroundDrawable(Drawable background)

不只可以设置背景，只有加上他，setOutsideTouchable()才会生效；popupWindow 才会对手机返回键有响应，点击手机返回键，可以关

闭 popupWindow，否则将直接关闭 activity。

(4)为 popupWindow 设置动画

生成动画对应的 style

2.ViewConfiguration

（1）Contains methods to standard constants used in the UI for timeouts, sizes, and distances.

包含 UI 超时设定，尺寸，距离等的静态常量方法

构造方法在过时，通过传递上下文对象 get()得到，取决于上下文对象的尺寸，密度等

（2）返回两次点击屏幕的间隔时间用来判断是否响应双击事件

（3）子控件被触摸持续时间

（4）第一次触屏和第二次触屏的像素间距，此操作仍被视为双击

（5）在我们判定操作时滑动之前，判定为点击的最大像素间距

5. Android 之自定义 view

5.1 基本步骤

自定义 view 属性

在 view 的构造方法中获得我们自定义的属性

重写 onMesure(非必须)

1. 自定义 view 属性，在 res/values/下新建 attrs.xml，在里面定义属性和申明整个样式。

format 控制属性的取值范围

string:字符串

color:颜色

dimension:尺寸大小

integer:整型数

enum:枚举数据

reference:引用资源

float:浮点型

boolean:布尔类型

fraction:百分数

flag:位运算

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="TitleView">
        <attr name="Text" format="string"/>
        <attr name="TextColor" format="color"/>
        <attr name="TextSize" format="dimension"/>
    </declare-styleable>
</resources>
```

2. 在布局文件中使用自定义 view

Android studio 中引入命名空间（eclipse 中需要指出工程包名）

```
xmlns:custom="http://schemas.android.com/apk/res-auto"
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:custom="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.ustb.shinerio.myview.MainActivity">
<com.ustb.shinerio.myview.TitleView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    custom:Text="3712"
    custom:TextColor="#FF00FA"
    custom:TextSize="40sp"
    />
</RelativeLayout>
```

3. 在 View 的构造方法中，获得我们的自定义的样式

```java
//java 代码中创建控件使用
public TitleView(Context context) {
    this(context, null);
}

//xml 布局中申明控件使用
public TitleView(Context context, AttributeSet set) {
    this(context, set, 0);
}
public TitleView(Context context, AttributeSet set, int defStyle) {
    super(context, set, defStyle);
    //获得自定义样式属性
    TypedArray typedArray = context.getTheme().obtainStyledAttributes(set,
R.styleable.TitleView, defStyle, 0);
    int n = typedArray.getIndexCount();
    for (int i = 0; i < n; i++) {
        int attr = typedArray.getIndex(i);
        switch (attr) {
            case R.styleable.TitleView_Text:
                    mTitleText = typedArray.getString(attr);
                break;
            case R.styleable.TitleView_TextColor:
                    mTitleTextColor = typedArray.getColor(attr, Color.BLACK);  //默认黑
色
                break;
            case R.styleable.TitleView_TextSize:
                //默认 16sp,利用 TypeValue 也可以把 sp 转化为 px
                    mTitleTextSize = typedArray.getDimensionPixelSize(attr,
```

```
                                (int)
TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_SP,16,getResources().getDisplayMetric
s()));
                    break;
            }
            typedArray.recycle();
            //获得绘制文本的宽和高
            mPaint = new Paint();
            mPaint.setTextSize(mTitleTextSize);  //设置绘图文本大小
            mBound = new Rect();
            //参数分别是，被用来测量约束限制的文本，被测量开始字符的索引，结束的索引，用来
返回被绑定的约束范围对象
            mPaint.getTextBounds(mTitleText,0,mTitleText.length(),mBound);
        }
```

4. 重写 onDraw()和 onMeasure()方法

```
@Override
protected void onDraw(Canvas canvas) {
    mPaint.setColor(Color.WHITE);
    canvas.drawRect(0,0,getMeasuredWidth(),getMeasuredHeight(),mPaint);    //绘制控件的
主体
    mPaint.setColor(mTitleTextColor);
    //设置文本的绘制位置，这里设置成居中，注意！！二参数是 x 起始坐标，而三坐标是 y 的基
准线！！！
    canvas.drawText(mTitleText,getWidth()/2-mBound.width()/2,
                    getHeight()/2+mBound.height()/2,mPaint);
}
```

不重写 onMeasure()会导致 wrap_content 铺满屏幕

MeasureSpec 的 specMode,一共三种类型：

EXACTLY：一般是设置了明确的值或者是 MATCH_PARENT

AT_MOST：表示子布局限制在一个最大值内，一般为 WARP_CONTENT

UNSPECIFIED：表示子布局想要多大就多大，很少使用

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    int widthMode = MeasureSpec.getMode(widthMeasureSpec);
    int widthSize = MeasureSpec.getSize(widthMeasureSpec);
    int heightMode = MeasureSpec.getMode(heightMeasureSpec);
```

```
    int heightSize = MeasureSpec.getSize(heightMeasureSpec);
    int width,height;
    if(widthMode == MeasureSpec.EXACTLY){
        width = widthSize;
    }else {
        width = getPaddingLeft()+getPaddingRight()+mBound.width();
    }
    if(heightMode == MeasureSpec.EXACTLY){
        height = heightSize;
    }else{
        height = getPaddingBottom()+getPaddingTop()+mBound.height();
    }
    setMeasuredDimension(width,height);
}
```

5.2 进阶

```
<attr name="image" format="reference"/>  <!--引用类型-->
<attr name="imageScaleType">             <!--枚举类型-->
    <enum name = "fillXY" value="0"/>
    <enum name = "center" value="1"/>
</attr>
```

自定义 view

```
public class CustomImageView extends View{
    private static final int FILLXY = 0;
    private static final int CENTER = 1;
    private String text="";
    private int textColor;
    private int textSize;
    private Bitmap bitmap;
    private int imageScaleType;
    private Rect mBound;
    private Paint mPaint;

    public CustomImageView(Context context) {
        this(context,null);
    }

    public CustomImageView(Context context, AttributeSet attrs){
        this(context,attrs,0);
    }
    public CustomImageView(Context context,AttributeSet attrs,int defStyle){
     super(context,attrs,defStyle);
```

```
        TypedArray typedArray =
context.getTheme().obtainStyledAttributes(attrs,R.styleable.CustomImageView,defStyle,0)
;
        //必要的属性，有默认值的，在 for 循环前初始化
        textColor =
typedArray.getColor(R.styleable.CustomImageView_TextColor,Color.BLACK);
        textSize =
typedArray.getDimensionPixelSize(R.styleable.CustomImageView_TextSize,

(int)TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_SP,16,getResources().getDisplayM
etrics()));
        bitmap =
BitmapFactory.decodeResource(getResources(),typedArray.getResourceId(R.styleable.Custom
ImageView_image,R.drawable.default_image));
        imageScaleType =
typedArray.getInt(R.styleable.CustomImageView_imageScaleType,0);
        int n = typedArray.getIndexCount();
        for(int i = 0;i<n;i++){
            int attr =typedArray.getIndex(i);
            switch (attr){
                case R.styleable.CustomImageView_Text:
                    text = typedArray.getString(attr);
                    break;
            }
        }
        typedArray.recycle();
        mPaint = new Paint();
        mBound = new Rect();
        //设置字体边界
        mPaint.getTextBounds(text,0,text.length(),mBound);
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        int widthMode = MeasureSpec.getMode(widthMeasureSpec);
        int heightMode = MeasureSpec.getMode(heightMeasureSpec);
        int widthSize = MeasureSpec.getSize(widthMeasureSpec);
        int heightSize = MeasureSpec.getSize(heightMeasureSpec);
        int width,height;
        if(widthMode==MeasureSpec.EXACTLY) {
            width = widthSize;
        }else {
            int imageWidth = getPaddingLeft()+getPaddingRight()+bitmap.getWidth();
            int textWidth = getPaddingLeft()+getPaddingRight()+mBound.width();
```

```
            //取最大
            width = Math.max(imageWidth,textWidth);
        }
        if(heightMode == MeasureSpec.EXACTLY){
            height = heightSize;
        }else{
            int imageHeight = getPaddingBottom()+getPaddingTop()+bitmap.getHeight();
            int textHeight = getPaddingBottom()+getPaddingTop()+mBound.height();
            height = Math.max(imageHeight,textHeight);
        }
        setMeasuredDimension(width,height);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        //先绘制边框
        mPaint.setStrokeWidth(2);
        mPaint.setStyle(Paint.Style.STROKE);  //只描边，不填充，FILL 是填充效果
        mPaint.setColor(Color.BLACK);
        canvas.drawRect(0,0,getMeasuredWidth(),getMeasuredHeight(),mPaint);
        //绘制文本
        mPaint.setTextSize(textSize);
        mPaint.setColor(textColor);
        if(mBound.width()>getMeasuredWidth()){
            //当设置文本大于指定的控件宽度时，设置字体为超过部分以...表示
            text = TextUtils.ellipsize(text,new TextPaint(),getMeasuredWidth()-
getPaddingRight()-getPaddingLeft(),TextUtils.TruncateAt.END).toString();
            canvas.drawText(text,getPaddingLeft(),getMeasuredHeight()-
getPaddingBottom(),mPaint);
        }else {
            //正常情况下文字居中
            canvas.drawText(text,getMeasuredWidth()/2-
mBound.width()/2,getMeasuredHeight()-getPaddingBottom(),mPaint);
        }
        //绘制图片
        if(imageScaleType== FILLXY){
            canvas.drawBitmap(bitmap,null,new
Rect(getPaddingLeft(),getPaddingTop(),getMeasuredWidth()-getPaddingRight(),
                    getMeasuredHeight()-getPaddingBottom()-mBound.height()),mPaint);
        }else {
            canvas.drawBitmap(bitmap,getMeasuredWidth()/2-bitmap.getWidth(),
                    (getMeasuredHeight()-mBound.height()-getPaddingBottom()-
getPaddingTop())/2+getPaddingTop()+bitmap.getHeight()/2
            ,mPaint);
```

```
        }
    }
}
```

5.3 Paint（画笔）和 Canvas(画布)

要绘制图形，首先得调整画笔，按照自己的开发需要设置画笔的相关属性。Pain 类的常用属性设置方法如下：

```
setAntiAlias();          //设置画笔的锯齿效果
setColor();              //设置画笔的颜色
setARGB();               //设置画笔的 A、R、G、B 值
setAlpha();              //设置画笔的 Alpha 值
setTextSize();           //设置字体的尺寸
setStyle();              //设置画笔的风格（空心或实心）
setStrokeWidth();        //设置空心边框的宽度
getColor();              //获取画笔的颜色
```

画笔属性设置好之后，还需要将图像绘制到画布上。Canvas 类可以用来实现各种图形的绘制工作，如绘制直线、矩形、圆等等。Canvas 绘制常用图形的方法如下：

绘制直线：canvas.drawLine(float startX, float startY, float stopX, float stopY, Paint paint);

绘制矩形：canvas.drawRect(float left, float top, float right, float bottom, Paint paint);

绘制圆形：canvas.drawCircle(float cx, float cy, float radius, Paint paint);

绘制字符：canvas.drawText(String text, float x, float y, Paint paint);

绘制图形：canvas.drawBirmap(Bitmap bitmap, float left, float top, Paint paint);