# 第3章 文字

## 065 使用 ScaleXSpan 创建扁平风格的文字

此实例主要通过使用 ScaleXSpan，实现在 TextView 中以扁平化的风格显示文本。当实例运行之后，在 TextView 中以扁平化的风格显示文本的效果如图 065.1 所示。


图 065.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
```

```
    TextView myTextView = (TextView) findViewById(R.id.myTextView);
    //创建一个 SpannableString 对象
    SpannableString mySpannableString = new SpannableString("实例集锦");
    //创建水平拉伸 3 倍的 ScaleXSpan
    mySpannableString.setSpan(new ScaleXSpan(3), 0,
            mySpannableString.length(), Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
    myTextView.setText(mySpannableString);
} }
```

上面这段代码在 MyCode\MySample040\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new ScaleXSpan(3)，0，mySpannableString.length()，Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)表示将指定的文字在水平方向拉伸 3 倍，0 和 mySpannableString.length()表示 ScaleXSpan 作用范围的起止位置。此实例的完整项目在 MyCode\MySample040 文件夹中。

## 066 使用 MaskFilterSpan 实现文字边缘模糊

此实例主要通过定制模糊滤镜 BlurMaskFilter 的模糊方式为 BlurMaskFilter.Blur.SOLID，在 TextView 中实现文本的背景模糊扩散。当实例运行之后，在 TextView 中文本的背景模糊扩散的效果如图 066.1 所示。


图 066.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    TextView myTextView = (TextView) findViewById(R.id.myTextView);
    SpannableStringBuilder mySpannableStringBuilder =
                                        new SpannableStringBuilder("炫酷");
    //定制滤镜的方式是 BlurMaskFilter.Blur.SOLID
    MaskFilterSpan myMaskFilterSpan = new MaskFilterSpan(
                          new BlurMaskFilter(21, BlurMaskFilter.Blur.SOLID));
    mySpannableStringBuilder.setSpan(myMaskFilterSpan, 0, 2,
                  Spanned.SPAN_INCLUSIVE_INCLUSIVE);       //设置滤镜的作用范围
    myTextView.setText(mySpannableStringBuilder);
} }
```

上面这段代码在 MyCode\MySample037\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，myMaskFilterSpan = new MaskFilterSpan(new BlurMaskFilter(21, BlurMaskFilter.Blur.SOLID))用于创建一个模糊距离是 21，模糊方式是 BlurMaskFilter.Blur.SOLID 的模糊滤镜。mySpannableStringBuilder.setSpan(myMaskFilterSpan, 0, 2, Spanned.SPAN_INCLUSIVE_INCLUSIVE)中的 0 表示模糊滤镜在字符串中发生作用的开始位置索引，2 表示模糊滤镜在字符串中发生作用的结束位置索引。此实例的完整项目在 MyCode\MySample037 文件夹中。

## 067 使用 MaskFilterSpan 实现文字中心镂空

此实例主要通过定制模糊滤镜 BlurMaskFilter 的模糊方式为 BlurMaskFilter.Blur.OUTER，实现在 TextView 中显示线条描边的镂空文本。当实例运行之后，在 TextView 中显示的线条描边的镂空文本的效果如图 067.1 所示。


图 067.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  SpannableStringBuilder mySpannableStringBuilder =
                                    new SpannableStringBuilder("炫酷");
  //定制滤镜的方式是 BlurMaskFilter.Blur.OUTER
  MaskFilterSpan myMaskFilterSpan = new MaskFilterSpan(
                        new BlurMaskFilter(1, BlurMaskFilter.Blur.OUTER));
  mySpannableStringBuilder.setSpan(myMaskFilterSpan, 0, 2,
                Spanned.SPAN_INCLUSIVE_INCLUSIVE);          //设置滤镜的作用范围
  myTextView.setText(mySpannableStringBuilder);
} }
```

上面这段代码在 MyCode\MySample036\app\src\main\java\com\bin\luo\mysample\ MainActivity.java 文件中。在这段代码中，myMaskFilterSpan = new MaskFilterSpan(new BlurMaskFilter(1, BlurMaskFilter.Blur.OUTER))用于创建一个模糊距离是 1，模糊方式是 BlurMaskFilter.Blur.OUTER 的模糊滤镜。mySpannableStringBuilder.setSpan(myMaskFilterSpan, 0, 2, Spanned.SPAN_INCLUSIVE_INCLUSIVE)中的 0 表示模糊滤镜在字符串中发生作用的开始位置索引，2 表示模糊滤镜在字符串中发生作用的结束位置索引。此实例的完整项目在 MyCode\MySample036 文件夹中。

## 068 使用 MaskFilterSpan 实现文字整体模糊

此实例主要通过使用 BlurMaskFilter，实现在 TextView 中模糊显示文本。当实例运行之后，在 TextView 中模糊显示文本的效果如图 068.1 所示。


图 068.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  SpannableStringBuilder mySpannableStringBuilder =
                                    new SpannableStringBuilder("炫酷");
  MaskFilterSpan myMaskFilterSpan = new MaskFilterSpan(
                      new BlurMaskFilter(15, BlurMaskFilter.Blur.NORMAL));
  mySpannableStringBuilder.setSpan(myMaskFilterSpan,
                                   0, 2, Spanned.SPAN_INCLUSIVE_INCLUSIVE);
  myTextView.setText(mySpannableStringBuilder);
} }
```

上面这段代码在 MyCode\MySample035\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，myMaskFilterSpan = new MaskFilterSpan(new BlurMaskFilter(15, BlurMaskFilter.Blur.NORMAL))用于设置模糊滤镜的模糊距离是 15，模糊方式是 BlurMaskFilter.Blur.NORMAL，除了 NORMAL 之外，常用的模糊方式还有：OUTER、INNER、SOLID。mySpannableStringBuilder.setSpan(myMaskFilterSpan，0，2，Spanned.SPAN_INCLUSIVE_INCLUSIVE)中的 0 表示模糊滤镜在字符串中发生作用的开始位置索引，2 表示模糊滤镜在字符串中发生作用的结束位置索引。此实例的完整项目在 MyCode\MySample035 文件夹中。

## 069 使用 MaskFilterSpan 模糊多个字符串

此实例主要通过使用 MaskFilterSpan，实现非连续的多个字符串以模糊效果显示。当实例运行之后，在“关键词：”输入框中输入“软件”，然后单击“开始搜索”按钮，则在下面的文本中，所有非“软件”的文本都将以模糊效果显示，如图 069.1 的左图所示。在“关键词：”输入框中输入“的”，然后单击“开始搜索”按钮，则在下面的文本中，所有非“的”的文本都将以模糊效果显示，如图 069.1 的右图所示。测试其他关键词将取得类似的效果。


图 069.1

主要代码如下：

```
public class MainActivity extends Activity {
 TextView myTextView;
 EditText myEditText;
 String myText;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  getWindow().getDecorView().setLayerType(View.LAYER_TYPE_SOFTWARE, null);
  setContentView(R.layout.activity_main);
  myEditText = (EditText) findViewById(R.id.myEditText);
  myTextView = (TextView) findViewById(R.id.myTextView);
  myText = myTextView.getText().toString();                    //记录原始文本内容
 }
 public void onClickButton(View v) {                           //响应单击"开始搜索"按钮
  //获取搜索字符串(关键词)
  String mySearch = myEditText.getText().toString();
  SpannableString mySpannableString = new SpannableString(myText);
  for (int i = 0; i < myText.length(); i++) {
   int start = i;                                              //记录起始索引值
   i = myText.indexOf(mySearch, i);                            //获取符合搜索结果的索引值
   if (i < 0) {                                                //未搜索到
    mySpannableString.setSpan(new MaskFilterSpan(new BlurMaskFilter(8,
            BlurMaskFilter.Blur.NORMAL)), start + mySearch.length() - 1, myText.
            length(), Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
    break;
   }
   //模糊显示不符合搜索结果的部分,以默认样式显示符合搜索结果的部分
   mySpannableString.setSpan(new MaskFilterSpan(new BlurMaskFilter(8,
           BlurMaskFilter.Blur.NORMAL)), start + mySearch.length() - 1,
           i, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
  }
  myTextView.setText(mySpannableString);                       //应用 SpannableString 对象
} }
```

上面这段代码在 MyCode\MySample799\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new MaskFilterSpan(new BlurMaskFilter(8, BlurMaskFilter.Blur.NORMAL)), start + mySearch.length() - 1, myText.length(), Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)表示对指定的文本进行模糊，范围从start+mySearch.length()-1 到 myText.length()，模糊半径为 8 像素。此实例的完整项目在MyCode\MySample799 文件夹中。

## 070 使用 BulletSpan 在文本首字前添加小圆点

此实例主要通过使用 BulletSpan，实现在 TextView 文本的首字前添加小圆点符号。当实例运行之后，在 TextView 文本的首字前添加小圆点符号的效果如图 070.1 所示。


图 070.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  SpannableString mySpannableString =
                          new SpannableString("炫酷应用实例集锦");
  //创建指定颜色的符号,并指定符号与后面文字之间宽度的 BulletSpan
  mySpannableString.setSpan(new BulletSpan(50,
                    Color.RED), 0, 1, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
  myTextView.setText(mySpannableString);
} }
```

上面这段代码在 MyCode\MySample039\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new BulletSpan(50,Color.RED), 0, 1, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)用于创建一个距离文本首字 50px 的红色小圆点。如果省略颜色值，则创建的小圆点的颜色与文本的颜色相同。此实例的完整项目在 MyCode\MySample039 文件夹中。

## 071 使用 StrikethroughSpan 添加文字删除线

此实例主要通过使用 StrikethroughSpan，实现在 TextView 中的部分文字上添加删除线。当实例运行之后，“便捷的”三个字的中间有一条删除线，如图 071.1 所示。

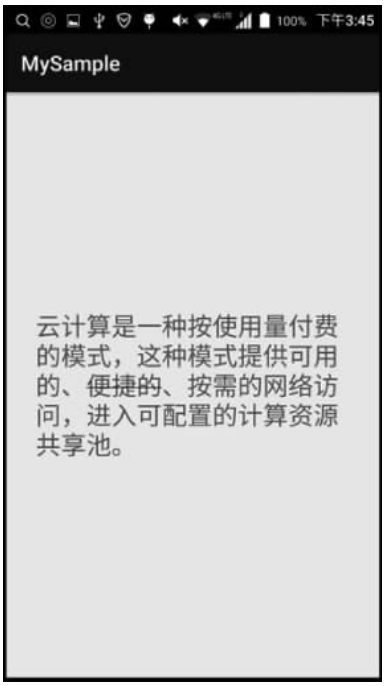


图 071.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  SpannableString mySpannableString = new SpannableString("云计算是一种按使用量付费的模式,这种模式
提供可用的、便捷的、按需的网络访问,进入可配置的计算资源共享池.");
  mySpannableString.setSpan(new StrikethroughSpan(), 26, 29,
    Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);        // 在"便捷的"上设置删除线
  myTextView.setText(mySpannableString);
} }
```

上面这段代码在 MyCode\MySample028\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new StrikethroughSpan()，26，29，Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)中的 26 和 29 指示删除线的范围，即“便捷的”三个字在字符串中的索引位置。如果需要为“便捷的”三个字添加下画线，则可以把代码修改为 myText.setSpan(new UnderlineSpan()，26，29，Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)。此实例的完整项目在 MyCode\MySample028 文件夹中。

## 072 使用 URLSpan 为部分内容添加超链接

此实例主要通过使用 URLSpan，实现在 TextView 中为部分内容添加超链接。当实例运行之后，“电话”“公司网站”和“公司地址”将以超链接的形式显示，如图 072.1 的左图所示；单击“电话”超链

接，则弹出手机的电话拨号盘；单击“公司网站”超链接，则打开网址指定的网站，如果手机上有多个浏览器，则显示浏览器选择窗口；单击“公司地址”超链接，则通过百度地图（默认地图工具）定位指定的地址，如图 072.1 的右图所示。


图 072.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  SpannableString mySpannableString = new SpannableString("我公司以国际化和专业化为目标,集聚了一批复合型、开拓型的营销和管理等各方面的人才,你可以通过电话与相关人员进行联系,也可以通过公司网站了解更多信息,还可以在这里查看公司地址.");
  //查找"电话"在字符串中的索引位置
  int myStartPos = mySpannableString.toString().indexOf("电话");
  int myEndPos = myStartPos + 2;
  mySpannableString.setSpan(new URLSpan("tel:13996060872"), myStartPos,
       myEndPos, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);          //响应单击"电话"超链接
  //查找"公司网站"在字符串中的索引位置
  myStartPos = mySpannableString.toString().indexOf("公司网站");
  myEndPos = myStartPos + 4;
  mySpannableString.setSpan(new URLSpan("http://www.baidu.com"), myStartPos,
    myEndPos, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);            //响应单击"公司网站"超链接
  //查找"公司地址"在字符串中的索引位置
  myStartPos = mySpannableString.toString().indexOf("公司地址");
  myEndPos = myStartPos + 4;
  //响应单击"公司地址"超链接
  mySpannableString.setSpan(new URLSpan("geo:38.899533,-77.036476"),
                    myStartPos, myEndPos, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
```

```
    myTextView.setText(mySpannableString);
    //实现单击超链接的功能
    myTextView.setMovementMethod(LinkMovementMethod.getInstance());
} }
```

上面这段代码在 MyCode\MySample033\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new URLSpan("tel:13996060872"), myStartPos, myEndPos, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)用于设置电话超链接，URLSpan("tel:13996060872")表示自动根据参数内容调用手机缺省应用，如果该参数是“tel:13996060872”，则调用电话拨号盘；如果该参数是“http://www.baidu.com”网址格式，则调用浏览器；如果该参数是“geo:38.899533,-77.036476”地址格式，则调用地图工具；myStartPos 表示超链接作用范围的开始位置，myEndPos 表示超链接作用范围的结束位置。但是仅设置了 URLSpan("tel:13996060872")，超链接不会起作用，必须调用 setMovementMethod(LinkMovementMethod.getInstance())方法，超链接才会发生作用。此实例的完整项目在 MyCode\MySample033 文件夹中。

## 073 使用 ImageSpan 同时显示 QQ 表情和文字

此实例主要通过使用 ImageSpan，实现在 TextView 中混合显示 QQ 表情和文字。当实例运行之后，在 TextView 中混合显示 QQ 表情和文字的效果如图 073.1 所示。


图 073.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    TextView myTextView = (TextView) findViewById(R.id.myTextView);
    SpannableString mySpannableString =
            new SpannableString("大爷,伤不起啊!我伤不起啊!伤不起啊!");
    //查找"我伤不起啊!"在字符串中的索引位置
    int myStartPos = mySpannableString.toString().indexOf("我伤不起啊!");
    int myEndPos = myStartPos + "我伤不起啊!".toString().length();
    //从 Drawable 获取 QQ 表情资源
    Drawable myDrawable = getResources().getDrawable(R.drawable.myqq);
    myDrawable.setBounds(0, 0, myDrawable.getIntrinsicWidth()/5,
            myDrawable.getIntrinsicHeight()/5);          //设置 QQ 表情图像的大小
    //使用 QQ 表情图像替换"我伤不起啊!"
    mySpannableString.setSpan(new ImageSpan(myDrawable),
                    myStartPos, myEndPos, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
    myTextView.setText(mySpannableString);
} }
```

上面这段代码在 MyCode\MySample034\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new ImageSpan(myDrawable)，myStartPos，myEndPos，Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)表示使用QQ表情图像替换文字“我伤不起啊!”，myStartPos 和 myEndPos 表示“我伤不起啊!”在整个字符串中的起止位置。此实例的完整项目在 MyCode\MySample034 文件夹中。

## 074 使用 StyleSpan 实现以粗斜体显示文字

此实例主要通过使用 StyleSpan，实现以粗斜体风格显示 TextView 的部分文字。当实例运行之后，“便捷的”三个字将以粗斜体风格显示，如图 074.1 所示。

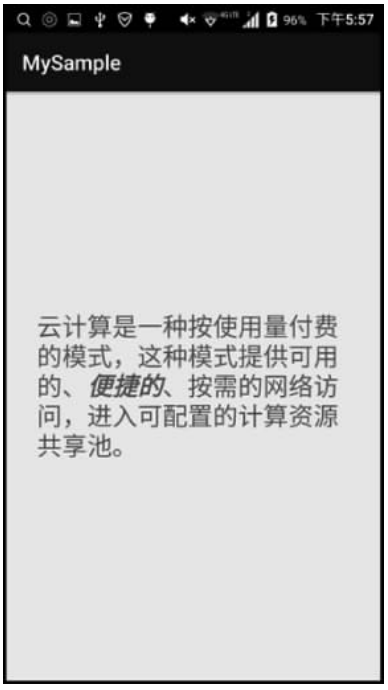


图 074.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  SpannableString mySpannableString = new SpannableString("云计算是一种按使用量付费的模式,这种模式
提供可用的、便捷的、按需的网络访问,进入可配置的计算资源共享池.");
  //以粗斜体风格显示"便捷的"三个字
  mySpannableString.setSpan(
                    new StyleSpan(android.graphics.Typeface.BOLD_ITALIC),
                    26, 29, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
  myTextView.setText(mySpannableString);
} }
```

上面这段代码在 MyCode\MySample029\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new StyleSpan(android.graphics.Typeface.BOLD_ITALIC),26,29, Spanned.SPAN_EXCLUSIVE_ EXCLUSIVE)中的 26 和 29 指示粗斜体的作用范围，即“便捷的”三个字在字符串中的索引位置；android. graphics.Typeface.BOLD_ITALIC 表示字体是粗斜体；如果此参数为 android.graphics. Typeface.ITALIC，则指定文字仅以斜体显示；如果此参数为 android.graphics.Typeface. BOLD，则指定文字仅以粗体显示。此实例的完整项目在 MyCode\MySample029 文件夹中。

## 075 使用 SuperscriptSpan 绘制勾股定理公式

此实例主要通过使用 SuperscriptSpan 和 RelativeSizeSpan，实现在 TextView 中以标准格式显示勾股定理公式。当实例运行之后，以标准格式显示勾股定理的效果如图 075.1 所示。

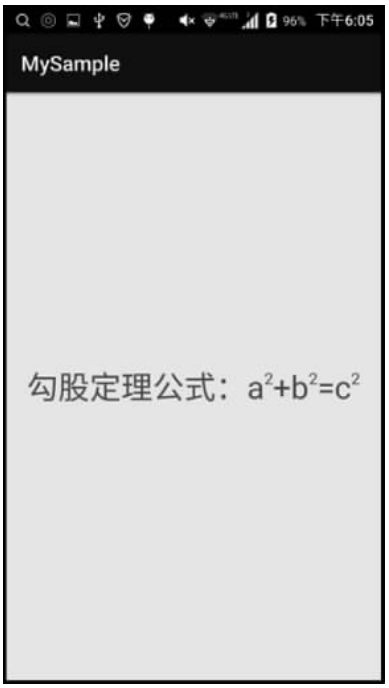


图 075.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  SpannableString mySpannableString =
                          new SpannableString("勾股定理公式:a2 + b2 = c2");
  //以上标的形式显示指数
  mySpannableString.setSpan(new SuperscriptSpan(),
                                8, 9, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
  mySpannableString.setSpan(new SuperscriptSpan(),
                                11, 12, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
  mySpannableString.setSpan(new SuperscriptSpan(),
                                14, 15, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
  // 0.5f 表示指数字体大小只有底数字体大小的一半
  mySpannableString.setSpan(new RelativeSizeSpan(0.5f),
                                8, 9, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
  mySpannableString.setSpan(new RelativeSizeSpan(0.5f),
                               11, 12, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
  mySpannableString.setSpan(new RelativeSizeSpan(0.5f),
                               14, 15, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
  myTextView.setText(mySpannableString);
} }
```

上面这段代码在 MyCode\MySample030\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new SuperscriptSpan(),8，9，Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)用于将指数 2 放在 a 的右上角，此时指数 2 的字体大小仍然与底数 a 相同；mySpannableString.setSpan(new RelativeSizeSpan(0.5f),8，9，Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)用于将指数 2 的字体大小缩小一半。在 Android 中，SuperscriptSpan 用于以上标的形式重置文本的显示位置，它一般用于数学公式中，RelativeSizeSpan 用于根据指定的参数缩放字体大小。此实例的完整项目在 MyCode\MySample030 文件夹中。

## 076 使用 SubscriptSpan 绘制硫酸亚铁分子式

此实例主要通过使用 SubscriptSpan 和 RelativeSizeSpan，实现在 TextView 中以下标形式显示硫酸亚铁的分子式。当实例运行之后，以下标形式显示的硫酸亚铁分子式的效果如图 076.1 所示。

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  SpannableString mySpannableString =
                          new SpannableString("硫酸亚铁的分子式:FeSO4");
  //查找 4 在字符串中的索引位置
  int myFePos = mySpannableString.toString().indexOf("4");
```

```
    //以下标的形式表示 FeSO4 中的 4
    mySpannableString.setSpan(new SubscriptSpan(),
                        myFePos, myFePos + 1, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
    //缩小下标字体的尺寸
    mySpannableString.setSpan(new RelativeSizeSpan(0.8f),
                        myFePos, myFePos + 1, Spanned. SPAN_EXCLUSIVE_EXCLUSIVE);
    myTextView.setText(mySpannableString);
  } }
```


图 076.1

上面这段代码在 MyCode\MySample031\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new SubscriptSpan(), myFePos, myFePos + 1, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)用于将 4 以下标的形式放在右下角，此时 4 的字体大小不变。myFePos 和 myFePos + 1 表示 SubscriptSpan()作用范围的开始和结束索引值。mySpannableString.setSpan(new RelativeSizeSpan(0.8f), myFePos, myFePos + 1,Spanned. SPAN_EXCLUSIVE_EXCLUSIVE)用于将 4 的字体大小缩小至原来的 4/5。在 Android 中，SubscriptSpan 用于以下标的形式重置文本的显示位置，它一般用于化学分子式中，RelativeSizeSpan 用于根据指定的参数缩放字体大小。此实例的完整项目在 MyCode\MySample031 文件夹中。

## 077 使用 TypefaceSpan 定制文本的部分内容

此实例主要通过使用 TypefaceSpan，实现对字符串中的部分文本定制个性化的字体。当实例运行之后，“微软风格的字体好看呀!”中的“微软”二字将以自定义字体显示，效果如图 077.1 所示。


图 077.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  myTextView.setText("微软风格的字体好看呀!");
  Typeface myTypeface = Typeface.createFromAsset(getAssets(),"myfont.ttf"); //加载指定字体
  myTextView.setTypeface(myTypeface);                                        //在 TextView 控件上应用
                                                                               该字体
  SpannableString mySpannableString = new SpannableString(myTextView.getText());
  //通过反向选取区域并应用字体,使其以默认字体样式显示
  mySpannableString.setSpan(new TypefaceSpan("myfont"), 2,
        myTextView.getText().length(), Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
  myTextView.setText(mySpannableString);                                     //生成效果并显示
} }
```

上面这段代码在 MyCode\MySample800\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new TypefaceSpan("myfont"), 2, myTextView.getText().length(), Spanned.SPAN_EXCLUSIVE _EXCLUSIVE)表示对 mySpannableString 字符串中的前 2 个字符使用字体“myfont”。需要说明的是，自定义字体文件 myfont.ttf 通常放在 MyCode\MySample800\app\src\main\ assets 目录中。此实例的完整项目在 MyCode\MySample800 文件夹中。

# 078 使用 ForegroundColorSpan 创建光照文字

此实例主要通过同时使用 ForegroundColorSpan 和 MaskFilterSpan，使字符串的部分文本呈现光照的浮雕特效。当实例运行之后，同时使用 ForegroundColorSpan 和 MaskFilterSpan 产生的部分文本(Google)浮雕特效如图 078.1 所示。


图 078.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  myTextView.setText("Google 谷歌");
  SpannableString mySpannableString = new SpannableString(myTextView.getText());
  mySpannableString.setSpan(new ForegroundColorSpan(Color.RED),
            0, 6, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);            //应用红色前景
  mySpannableString.setSpan(new MaskFilterSpan(
          new EmbossMaskFilter(new float[]{1, 1, 1}, 0.25f, 6, 3.5f)),
          0, 6, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);            //应用光照特效
  myTextView.setText(mySpannableString);
} }
```

上面这段代码在 MyCode\MySample801\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，mySpannableString.setSpan(new ForegroundColorSpan(Color.RED)，0，6，Spanned.SPAN_EXCLUSIVE_EXCLUSIVE)表示对 mySpannableString 字符

串的前 6 个字符(“Google”)应用红色前景。mySpannableString. setSpan (new MaskFilterSpan(new EmbossMaskFilter(new float[]{1, 1, 1}, 0.25f, 6, 3.5f)), 0, 6, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)表示对 mySpannableString 字符串的前 6 个字符应用 EmbossMaskFilter 产生的光照特效。在测试此实例时,通常应该关闭硬件加速,即设置 android: hardwareAccelerated="false",该代码在 MyCode\MySample801\ app\src\main\AndroidManifest.xml 文件中。此实例的完整项目在 MyCode\MySample801 文件夹中。

## 079 使用 BlurMaskFilter 创建阴影扩散文字

此实例主要通过在自定义 View 中创建 BlurMaskFilter 模糊滤镜,并在画笔 Paint 中使用 setMaskFilter()方法设置模糊滤镜,实现绘制阴影扩散的文字。当实例运行之后,在自定义 View 中使用 BlurMaskFilter 模糊滤镜绘制阴影扩散的文字的效果如图 079.1 所示。

主要代码如下:

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  setContentView(new MyView(this));
  super.onCreate(savedInstanceState);
}
class MyView extends View {
  private Paint myPaint;
  public MyView(Context context) {
   super(context);
   myPaint = new Paint();
   myPaint.setFlags(Paint.ANTI_ALIAS_FLAG);
   myPaint.setAntiAlias(true);
   myPaint.setColor(Color.BLACK);
   myPaint.setTextSize(220);
   myPaint.setStyle(Paint.Style.FILL_AND_STROKE);
   myPaint.setStrokeWidth(2);
   BlurMaskFilter myBlurMaskFilter =
        new BlurMaskFilter(40, BlurMaskFilter.Blur.SOLID);
                                          //创建模糊滤镜
   myPaint.setMaskFilter(myBlurMaskFilter);   //设置模糊滤镜画笔
  }
  @Override
  protected void onDraw(Canvas myCanvas) {
   super.onDraw(myCanvas);
   Display myDisplay = getWindowManager().getDefaultDisplay();
   int myWidth = myDisplay.getWidth();
   int myHeight = myDisplay.getHeight();
   myCanvas.drawText("炫酷实例", myWidth/10,
                     myHeight/2 - 150, myPaint); //显示扩散阴影文本
}}}
```


图 079.1

上面这段代码在 MyCode\MySample087\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中,myBlurMaskFilter = new BlurMaskFilter(40, BlurMaskFilter.Blur.SOLID)用于创建模糊滤镜,40 表示模糊半径,BlurMaskFilter.Blur.SOLID 表示模糊模式是 SOLID。BlurMaskFilter 支持下列 4 种模糊模式。

(1) NORMAL,表示同时绘制图形本身内容+内阴影+外阴影,正常阴影效果。
(2) INNER,表示绘制图形内容本身+内阴影,不绘制外阴影。
(3) OUTER,表示不绘制图形内容以及内阴影,只绘制外阴影。
(4) SOLID,表示只绘制外阴影和图形内容本身,不绘制内阴影。
此实例的完整项目在 MyCode\MySample087 文件夹中。

## 080 使用 EmbossMaskFilter 创建浮雕文字

此实例主要通过使用浮雕滤镜 EmbossMaskFilter,实现在 TextView 中创建浮雕风格的文本。当实例运行之后,在 TextView 中显示的浮雕文本的效果如图 080.1 所示。


图 080.1

主要代码如下:

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  SpannableStringBuilder mySpannableStringBuilder =
          new SpannableStringBuilder("炫酷");
   float[] myDirection = new float[]{ 10, 10, 10 };          //设置光源的方向
   float myLight = 0.1f;                                     //环境光亮度
   float mySpecular = 5;                                     //反射等级
   float myBlur = 5;                                         //模糊半径
   //根据指定的参数创建浮雕滤镜 EmbossMaskFilter
   EmbossMaskFilter myEmbossMaskFilter =
```

```
            new EmbossMaskFilter(myDirection,myLight,mySpecular,myBlur);
    //根据浮雕滤镜创建 MaskFilterSpan
     MaskFilterSpan myMaskFilterSpan = new MaskFilterSpan(myEmbossMaskFilter);
     mySpannableStringBuilder.setSpan(myMaskFilterSpan,0, 2,
        Spanned.SPAN_INCLUSIVE_INCLUSIVE);                    //设置 MaskFilterSpan 的作用范围
    myTextView.setText(mySpannableStringBuilder);
} }
```

上面这段代码在 MyCode\MySample038\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，myEmbossMaskFilter = new EmbossMaskFilter(myDirection，myLight，mySpecular，myBlur）用于根据指定的参数 myDirection，myLight，mySpecular，myBlur 创建浮雕滤镜；当浮雕滤镜创建后，再使用该浮雕滤镜创建 MaskFilterSpan，即 myMaskFilterSpan = new MaskFilterSpan（myEmbossMaskFilter）。mySpannableStringBuilder.setSpan(myMaskFilterSpan，0，2，Spanned.SPAN_INCLUSIVE_ INCLUSIVE)中的 0 表示浮雕滤镜发生作用的开始位置索引，2 表示浮雕滤镜发生作用的结束位置索引。在测试此实例时，通常应该关闭硬件加速，即在 MyCode\ MySample038\app\src \main\AndroidManifest.xml 文件中设置 android：hardwareAccelerated="false"。此实例的完整项目在 MyCode\MySample038 文件夹中。

## 081 通过自定义 View 在半圆弧上绘制文字

此实例主要通过使用 Path 的 addArc()方法和 Canvas 的 drawTextOnPath()方法，在自定义 View 中实现在半圆弧路径上绘制文字。当实例运行之后，在半圆弧路径上绘制的文字“Android 炫酷应用实例集锦”的效果如图 081.1 所示。


图 081.1

主要代码如下：

```
<com.bin.luo.mysample.CustomTextView android:layout_width="match_parent"
                                     android:layout_height="wrap_content"/>
```

上面这段代码在 MyCode\MySample775\app\src\main\res\layout\activity_main.xml 文件中。在这段代码中，com.bin.luo.mysample.CustomTextView 是自定义 View 类（控件）在布局文件中的应用，com.bin.luo.mysample 是包名，在实际应用中通常应该修改为自己的包名，自定义类 CustomTextView 的主要代码如下：

```
public class CustomTextView extends View {
 String myText = "Android 炫酷应用实例集锦";
 Paint myPaint = null;
 Path myPath = null;
 public CustomTextView(Context context, AttributeSet attrs) {
  super(context, attrs);
  Init();
 }
 public void Init() {                                          //初始化画笔和路径对象
  myPaint = new Paint();
  myPaint.setAntiAlias(true);
  myPaint.setTextSize(96);
  LinearGradient myGradient = new LinearGradient(100,100, 800, 800,
       Color.GREEN, Color.BLUE, Shader.TileMode.MIRROR);       //初始化线性渐变对象
  myPaint.setShader(myGradient);                               //使用渐变色填充画笔颜色
  if (myPath == null) {                                        //若未设置路径对象,则默认将以半圆路径显
                                                                 示文本
   myPath = new Path();
   myPath.addArc(100,100, 800, 800,180, 180);                  //绘制半圆路径
 } }
 @Override
 protected void onDraw(Canvas canvas) {
  super.onDraw(canvas);
  canvas.translate(100,490);                                   //平移画布至指定位置
  canvas.drawTextOnPath(myText, myPath, 3,
                                  - 10, myPaint);              //在指定路径上绘制指定文本
} }
```

上面这段代码在 MyCode\MySample775\app\src\main\java\com\bin\luo\mysample\CustomTextView.java 文件中。在这段代码中，myGradient = new LinearGradient(100，100，800，800，Color.GREEN，Color.BLUE，Shader.TileMode.MIRROR)用于创建一个填充 Paint 的线性渐变色对象，LinearGradient()构造函数的语法声明如下：

```
public LinearGradient(float x0, float y0, float x1,
                      float y1, int color0, int color1, TileMode tile)
```

其中，参数 float x0 表示渐变起点的 $x$ 坐标；参数 float y0 表示渐变起点的 $y$ 坐标；参数 float x1 表示渐变终点的 $x$ 坐标；参数 float y1 表示渐变终点的 $y$ 坐标；参数 int color0 表示渐变开始颜色；参数 int color1 表示渐变结束颜色；参数 TileMode tile 表示平铺方式。TileMode 有 3 种参数可供选择，分别为 CLAMP、REPEAT 和 MIRROR；CLAMP 的作用是如果渲染器超出原始边界范围，则会复制边缘颜色对超出范围的区域进行着色，REPEAT 的作用是在横向和纵向上以平铺的形式重复渲染，MIRROR 的作用是在横向和纵向上以镜像的方式重复渲染。

myPath.addArc(100,100, 800, 800,180, 180)的作用是绘制一个半圆弧路径。addArc()方法的语法声明如下：

```
public void addArc(float left, float top,
                float right, float bottom, float startAngle, float sweepAngle)
```

其中，参数 float left 表示圆弧外接矩形左上角的 $x$ 坐标，参数 float top 表示圆弧外接矩形左上角的 $y$ 坐标，参数 float right 表示圆弧外接矩形右下角的 $x$ 坐标，参数 float bottom 表示圆弧外接矩形右下角的 $y$ 坐标，float startAngle 表示圆弧起始角，此角度不好理解，它以水平线右端为 0°，以顺时针方向为增量，此实例所用的 180°即在水平线的左端，float sweepAngle 表示圆弧的度数范围，在此实例中，startAngle 为 180，sweepAngle 为 180，因此终止角为 360°，即水平线的右端。如果 myPath.addArc(9, 9, 350, 350,90,180)，则圆弧文字表现为左端，而不是上端。

canvas.drawTextOnPath(myText, myPath, 3, -10, myPaint)表示在指定的圆弧路径上绘制渐变色文字。drawTextOnPath()方法的语法声明如下：

```
public void drawTextOnPath(@NonNull String text, @NonNull Path path,
                        float hOffset, float vOffset, @NonNull Paint paint)
```

其中，参数 String text 表示显示的文字，参数 Path path 表示文字的显示路径，参数 float hOffset 表示水平偏移量，参数 float vOffset 表示垂直偏移量，参数 Paint paint 表示定制的画笔。

此实例的完整项目在 MyCode\MySample775 文件夹中。

## 082 通过自定义 View 在圆弧上滚动文字

此实例主要通过使用 Timer 定时修改 drawTextOnPath()方法的 hOffset 参数(水平偏移量)，实现使文字沿着自定义圆弧路径滚动显示。当实例运行之后，单击“开始滚动”按钮，则演示文本“人生得意须尽欢”将沿着自定义圆弧路径以顺时针方向滚动显示；单击“暂停滚动”按钮，则滚动显示停止；再次单击“开始滚动”按钮，则演示文本将从上次暂停的位置继续滚动显示，效果分别如图 082.1 的左图和右图所示。


图 082.1

主要代码如下：

```
//响应单击"开始滚动"按钮
public void onClickBtn1(View v){ myCustomTextView.startMarqueeAnim(); }
 //响应单击"暂停滚动"按钮
public void onClickBtn2(View v){ myCustomTextView.pauseMaruqeeAnim(); }
```

上面这段代码在 MyCode\MySample776\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，myCustomTextView = (CustomTextView) findViewById(R.id.myCustomTextView)中的 CustomTextView 是自定义控件(类)，该控件(类)的主要代码如下：

```
public class CustomTextView extends View {
 String myText = "人生得意须尽欢";
 Paint myPaint = null;
 Path myPath = null;
 Timer myTimer;
 TimerTask myTask;
 float myPositionX = 0;
 public CustomTextView(Context context, AttributeSet attrs) {
  super(context, attrs);
  Init();
 }
 public void Init() {                          //初始化画笔和路径对象
  myPaint = new Paint();
  myPaint.setAntiAlias(true);
  myPaint.setTextSize(80);
  if (myPath == null) {                        //若未设置路径对象,则默认将以半圆路径显示文本
   myPath = new Path();
   myPath.addArc(10, 10, 700, 700, 0, 360);   //绘制半圆路径
  } }
 @Override
 protected void onDraw(Canvas canvas) {
  super.onDraw(canvas);
  canvas.translate(150, 300);                  //平移画布至指定位置
  canvas.drawTextOnPath(myText, myPath,
    myPositionX, -10, myPaint);                //在指定路径上绘制指定文本
 }
  public void startMarqueeAnim() {            //创建定时任务
  if (myTask != null) { myTask.cancel(); }
  myTimer = new Timer();
  myTask = new CustomTimerTask();             //初始化自定义定时任务类
  myTimer.schedule(myTask, 0, 100);           //提交任务至定时器并开始执行
 }
 //取消定时任务
 public void pauseMaruqeeAnim() { myTask.cancel(); }
 private class CustomTimerTask extends TimerTask {
  @Override
  public void run() {
   myPositionX += 10;
   if (myPositionX == 1500) { myPositionX = 0; }
   postInvalidate();
} } }
```

上面这段代码在 MyCode\MySample776\app\src\main\java\com\bin\luo\mysample\

CustomTextView.java 文件中。在这段代码中，myTimer.schedule(myTask, 0, 100)表示间隔执行滚动显示文本任务，schedule()方法的语法声明如下：

```
public void schedule(TimerTask task, long delay, long period)
```

其中，参数 TimerTask task 是一个 TimerTask 派生类的实例，该派生类通常需要实现 public void run()方法，因为 TimerTask 类实现了 Runnable 接口。参数 long delay 用于设置 timer 定时器第一次调用 run 方法的时间，0 表示不设置时间，立刻执行任务。参数 long period 表示第一次执行任务之后，从第二次开始每隔多长的时间调用一次 run()方法，以 ms 为单位。

此实例的完整项目在 MyCode\MySample776 文件夹中。

## 083 通过自定义 View 绘制渐变色的文字

此实例主要通过使用线性渐变 LinearGradient 创建着色器 Shader，并在画笔 Paint 中使用 setShader()方法设置 Shader，实现绘制颜色渐变的文字。当实例运行之后，在自定义 View 中绘制的颜色渐变文字的效果如图 083.1 所示。


图 083.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  setContentView(new MyView(this));
  super.onCreate(savedInstanceState);
 }
```

```
class MyView extends View {
  public MyView(Context context) { super(context); }
  @Override
  protected void onDraw(Canvas myCanvas) {
   Display myDisplay = getWindowManager().getDefaultDisplay();
   int myWidth = myDisplay.getWidth();                    //获取屏幕宽度
   int myHeight = myDisplay.getHeight();                  //获取屏幕高度
   Shader myShader = new LinearGradient(0, 0, 160, 160,
           new int[]{Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW},
           null, Shader.TileMode.REPEAT);                 //创建渐变色的着色器
   Paint myPaint = new Paint();
   myPaint.setShader(myShader);                           //使用渐变色着色器设置画笔
   myPaint.setTextSize(160);                              //设置文字大小
   myCanvas.drawColor(Color.WHITE);                       //设置背景颜色
   myCanvas.drawText("炫酷应用实例", 50,
                         myHeight / 2 - 200, myPaint);  //绘制渐变色文字
    }
  }
}
```

上面这段代码在 MyCode\MySample085\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，myShader = new LinearGradient(0, 0, 160, 160, new int[]{Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW}, null, Shader.TileMode.REPEAT)用于创建渐变色的着色器。myCanvas.drawText("炫酷应用实例", 50, myHeight / 2 − 200, myPaint)中的 50 表示绘制文字开始位置的水平坐标，myHeight / 2 − 200 表示绘制文字开始位置的垂直坐标。此实例的完整项目在 MyCode\MySample085 文件夹中。

## 084 通过自定义 View 绘制线条描边文字

此实例主要通过设置画笔的样式为 STROKE，实现在自定义 View 中绘制线条描边的空心文字。当实例运行之后，在自定义 View 中绘制的线条描边的空心文字的效果如图 084.1 所示。

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  setContentView(new MyView(this));
  super.onCreate(savedInstanceState);
 }
 class MyView extends View {
  private Paint myPaint;
  public MyView(Context context) {
   super(context);
   myPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
   myPaint.setColor(Color.BLUE);                  //设置文字颜色
   myPaint.setStyle(Paint.Style.STROKE);          //创建空心文字
   myPaint.setStrokeWidth(1);                     //设置文字线条宽度
   myPaint.setTextSize(220);                      //设置字体大小
  }
  @Override
  protected void onDraw(Canvas myCanvas) {
   super.onDraw(myCanvas);
   Display myDisplay = getWindowManager().getDefaultDisplay();
   int myWidth = myDisplay.getWidth();
```


图 084.1

```
        int myHeight = myDisplay.getHeight();
        myCanvas.scale(1.0f,1.5f);                    //在垂直方向拉伸 1.5 倍
        myCanvas.drawText("炫酷实例", myWidth/10, myHeight * 3/10, myPaint);
      } }
    }
```

上面这段代码在 MyCode\MySample088\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，myPaint.setStyle(Paint.Style.STROKE)用于设置画笔的样式为空心。在 Android 中，Paint.Style 用于设置画笔的样式，取值如下。

(1) FILL，该样式用于创建实心画笔。

(2) FILL_AND_STROKE，该样式用于使画笔同时实现实心和空心效果，该样式在某些场合会带来不可预期的显示效果。

(3) STROKE，该样式用于创建空心画笔。

此实例的完整项目在 MyCode\MySample088 文件夹中。

## 085 通过自定义 View 绘制阴影扩散文字

此实例主要通过使用 setShadowLayer()方法，实现在自定义 View 中绘制阴影扩散的文字。当实例运行之后，在自定义 View 中绘制的阴影扩散文字的效果如图 085.1 所示。

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  setContentView(new MyView(this));
  super.onCreate(savedInstanceState);
 }
 class MyView extends View {
  private Paint myPaint;
  public MyView(Context context) {
   super(context);
   myPaint = new Paint();
   myPaint.setFlags(Paint.ANTI_ALIAS_FLAG);
   myPaint.setAntiAlias(true);
   myPaint.setColor(Color.BLUE);
   myPaint.setTextSize(200);
   myPaint.setStyle(Paint.Style.FILL_AND_STROKE);
   myPaint.setStrokeWidth(2);
   myPaint.setShadowLayer(15f, 10f, 10f, Color.GRAY);     //设置阴影画笔
  }
  @Override
  protected void onDraw(Canvas myCanvas) {
   super.onDraw(myCanvas);
   Display myDisplay = getWindowManager().getDefaultDisplay();
   int myWidth = myDisplay.getWidth();
   int myHeight = myDisplay.getHeight();
   myCanvas.drawText("炫酷实例", myWidth / 10,
                              myHeight * 2 / 5, myPaint); //绘制阴影文字
} } }
```


图 085.1

上面这段代码在 MyCode\MySample091\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，myPaint.setShadowLayer(15f,10f,10f, Color.GRAY)用于设置阴影画笔，15f 表示扩散半径，10f 表示水平方向的偏移量，10f 表示垂直方向的偏移量，Color.GRAY 表示阴影颜色。此实例的完整项目在 MyCode\MySample091 文件夹中。

## 086 加载字库文件显示自定义草书字体

此实例主要通过使用 Typeface 的 createFromAsset()方法根据指定的草体字库文件创建自定义字体，实现以草体字显示文本。当实例运行之后，单击“以默认字体显示”按钮，则唐诗将以默认的黑体字显示，如图 086.1 的左图所示。单击“以草体显示”按钮，则唐诗将以自定义的草体字显示，如图 086.1 的右图所示。

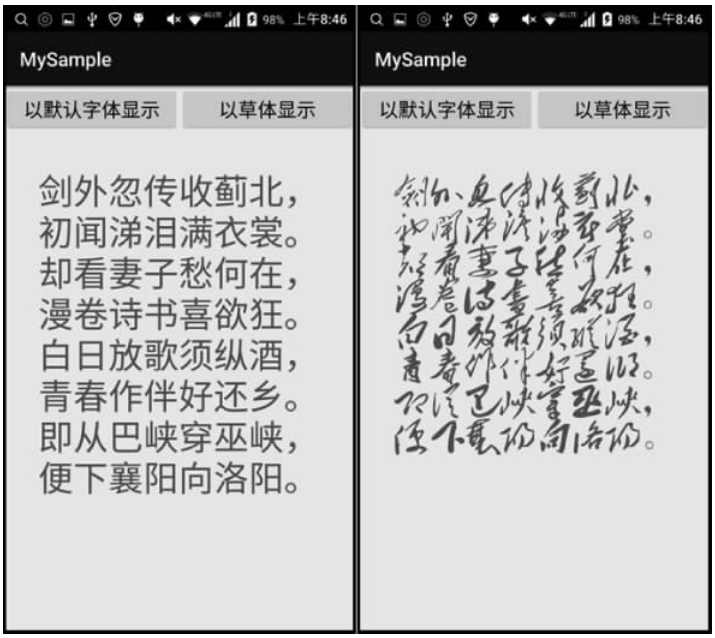


图 086.1

主要代码如下：

```
//响应单击"以默认字体显示"按钮
public void onClickmyBtn1(View v){ myTextView.setTypeface(Typeface.DEFAULT);}
//响应单击"以草体显示"按钮
public void onClickmyBtn2(View v) {
  Typeface myFont = Typeface.createFromAsset(this.getAssets(), "myfont.ttf");
  myTextView.setTypeface(myFont);          //加载自定义草体字
}
```

上面这段代码在 MyCode\MySample644\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，myFont = Typeface.createFromAsset(this.getAssets(), "myfont.ttf")用于根据草体字库文件 myfont.ttf 创建草体字。需要说明的是，如果当前项目不存在 assets 子目录，应该首先在 app\src\main 目录下创建 assets 子目录，然后在 assets 子目录中添加草体字库文件 myfont.ttf。此实例的完整项目在 MyCode\MySample644 文件夹中。

# 087 加载字库文件显示自定义液晶字体

此实例主要通过在 assets 目录中添加自定义液晶字体，并使用 createFromAsset()方法获取此字体，实现在 TextView 中显示自定义的液晶字体文字。当实例运行之后，“HTML AND CSS”以液晶字体显示的效果如图 087.1 所示。


图 087.1

主要代码如下：

```
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView myTextView = (TextView) findViewById(R.id.myTextView);
  AssetManager myAssetManager = getAssets();                    //获取 AssetManager
  Typeface myTypeface = Typeface.createFromAsset(myAssetManager,
                    "myLed.TTF");                               //根据液晶字体路径获取 Typeface
  myTextView.setTypeface(myTypeface);                           //设置当前字体为液晶字体
} }
```

上面这段代码在 MyCode\MySample019\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，setTypeface(myTypeface)用于设置 myTypeface 字体为当前文字的字体，myTypeface = Typeface.createFromAsset(myAssetManager, "myLed.TTF")则用于将 myLed.TTF 字体文件以资源的形式导入字体库。此实例的完整项目在 MyCode\MySample019 文件夹中。

# 088 判断在一个字符串中是否包含汉字

此实例主要通过使用 Pattern 和 Matcher 的成员方法，实现根据正则表达式“[\u4E00－\u9FA5\uF900－\uFA2D]”判断在一个字符串中是否包含汉字。当实例运行之后，如果在“测试内容：”输入框中输入“China 是一个伟大的国家”，然后单击“检测该字符串是否包含汉字”按钮，则在弹出的 Toast 中提示“该字符串有汉字!”，如图 088.1 的左图所示。如果在“测试内容：”输入框中输入“China is a great country”，然后单击“检测该字符串是否包含汉字”按钮，则在弹出的 Toast 中提示“该字符串没有汉字!”，如图 088.1 的右图所示。


图 088.1

主要代码如下：

```
public void onClickmyBtn1(View v) {          //响应单击"检测该字符串是否包含汉字"按钮
  String myText = myEditText.getText().toString();
  final String format = "[\\u4E00-\\u9FA5\\uF900-\\uFA2D]";
  Pattern myPattern = Pattern.compile(format);
  Matcher myMatcher = myPattern.matcher(myText);
  boolean myResult = myMatcher.find();
  if(myResult){
   Toast.makeText(getApplicationContext(),
                       "该字符串有汉字!", Toast.LENGTH_SHORT). show();
  }else{
   Toast.makeText(getApplicationContext(),
                       "该字符串没有汉字!", Toast.LENGTH_SHORT). show();
} }
```

上面这段代码在 MyCode\MySample539\app\src\main\java\com\bin\luo\mysample\MainActivity.java 文件中。在这段代码中，myMatcher.find()的返回值如果为 true，表示测试内容与正则表达式匹配；如果为 false，表示测试内容与正则表达式不匹配。此实例的完整项目在 MyCode\MySample539 文件夹中。