# 1,viewflow

一般情况下,当你需要做一个滑动然而不确定view的数目时,可以考虑使用ViewFlow。如果你的view数目确定,使用Fragments 或兼容库里的ViewPager比较好 。viewflow 是控件的源代码，使用的时候将其中复制到自己的项目中,还有values文件夹中的attr这个文件一起复制。

类库中主要包含的组件如下：

|  |  |
| --- | --- |
| Viewflow | 页面分屏滑动的组件 |
| CircleFlowIndicator | 圆点样式的当前页面指示器 |
| titleFlowindicator | 文本标题样式的当前页面指示器 |

## 1.1 CircleFlowIndicator

布局文件使用Viewflow和Flowindicator：

|  |
| --- |
| <FrameLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  xmlns:app=[*http://schemas.android.com/apk/res/org.taptwo.android.widget.viewflow.example*](http://schemas.android.com/apk/res/org.taptwo.android.widget.viewflow.example) *//需要引用应用的包命名空间*  android:orientation=*"vertical"* android:layout\_width=*"fill\_parent"*  android:layout\_height=*"fill\_parent"*>  <org.taptwo.android.widget.CircleFlowIndicator  android:padding=*"10dip"*  android:layout\_height=*"wrap\_content"*  android:layout\_width=*"wrap\_content"*  android:id=*"@+id/viewflowindic"*  android:layout\_gravity=*"center\_horizontal"*  app:inactiveType=*"fill"*  app:fadeOut=*"1000"* />  <org.taptwo.android.widget.ViewFlow  android:id=*"@+id/viewflow"* android:layout\_width=*"fill\_parent"*  android:layout\_height=*"fill\_parent"*  app:sidebuffer=*"3"*>  </org.taptwo.android.widget.ViewFlow>  </FrameLayout> |

Activity代码：

|  |
| --- |
| viewFlow = (ViewFlow) findViewById(R.id.*viewflow*);  viewFlow.setAdapter(**new** ImageAdapter(**this**), 5);//设置要显示的组件和默认显示的页面  CircleFlowIndicator indic = (CircleFlowIndicator) findViewById(R.id.*viewflowindic*);  viewFlow.setFlowIndicator(indic);//设置页面提示器 |

## 1.2 titleFlowindicator

文本标题的提示器需要指定要显示的文本，其他代码和布局文件没有大的变化

布局文件：

|  |
| --- |
| <org.taptwo.android.widget.TitleFlowIndicator  android:id=*"@+id/viewflowindic"*  android:layout\_height=*"wrap\_content"*  android:layout\_width=*"fill\_parent"*  app:footerLineHeight=*"2dp"*  app:customTypeface=*"fonts/Antic.ttf"*  app:footerTriangleHeight=*"10dp"*  app:textColor=*"#FFFFFFFF"* app:selectedColor=*"#FFFFC445"*  app:footerColor=*"#FFFFC445"* app:titlePadding=*"10dp"*  app:textSize=*"11dp"*  app:selectedSize=*"12dp"*  android:layout\_marginTop=*"10dip"*  app:clipPadding=*"5dp"*></org.taptwo.android.widget.TitleFlowIndicator> |

Activity代码：

|  |
| --- |
| viewFlow = (ViewFlow) findViewById(R.id.*viewflow*);  AndroidVersionAdapter adapter = **new** AndroidVersionAdapter(**this**);  viewFlow.setAdapter(adapter, 3);  TitleFlowIndicator indicator = (TitleFlowIndicator)findViewById(R.id.*viewflowindic*);  indicator.setTitleProvider(adapter);  viewFlow.setFlowIndicator(indicator); |

从上面代码中可以看出，viewflow和TitleFlowIndicator是使用的同一个adapter，这样才可以达到页面和指示器同时变化的效果。Adapter需要在继承baseAdapter的基础上实现TitleProvider接口。baseAdapter的getView方法用于返回viewflow要显示的组件或布局；TitleProvider的getTitle用于返回指示器要显示的文本，与viewflow要显示的组件是一一对应的。

|  |
| --- |
| **public** **class** AndroidVersionAdapter **extends** BaseAdapter **implements** TitleProvider  例子中完整的adapter代码如下： |

# 2，GreenDroid

greenDroid是一个android的UI开发类库，可以使你的UI开发更加简便，源码中包含一个类库和一个示例程序，做出来的效果很不错。

greenDroid导入工程后悔出现getActionbar方法问题，是因为greenDroid为了兼容android3之前的版本，自定义实现了ActionBar，与android 3之后自带actinbar有冲突，需要按照以下方式修改：<http://m.blog.csdn.net/blog/arest/19609591>

## 2.1 listview中的Item

GreenDroid中包含很多的Item，直接实例化这些Item并添加到ItemAdapter就可以作为listview项显示。下面介绍常用的几个Item类：

* **DescriptionItem**：

——用于描述的显示项，可以在界面中管显示多行文本

|  |
| --- |
| **new** DescriptionItem("仅用于显示文本");  与DescriptionItem类似的还有**TestItem，LongTestItem，SubTitleItem，SubTextItem**，也是用于显示文字，不同的是比DescriptionItem的默认字体要大，其他用法一样 |

* **SeparatorItem**：

——用于分割显示项，同时具有显示文本，类似系统设置中不同设置项分类

|  |
| --- |
| **new** SeparatorItem("分割显示区域的文本"); |

* **ThumbnailItem**：

——组合样式显示，左侧图片，右侧同时包含主标题和副标题，不同字号显示

|  |
| --- |
| **new** ThumbnailItem("主标题", "副标题", R.drawable.*class1*);//图片 |

* **DrawableItem**:

——组合样式显示，包含左侧图片和右侧的标题（没有副标题）

|  |
| --- |
| **new** DrawableItem("右侧的标题", R.drawable.*class2*);//图片 |

* **ProgressItem**：

——组合样式显示，左侧环形进度条，右侧为提示文本

|  |
| --- |
| progressItem = **new** ProgressItem("正在运行，请稍等", **true**); |

除了上面可以在代码中创建对象添加到ItemAdapter中，还可以在R.xml的xml文件中定义要显示的内容，然后解析添加到Adapter。另外每个Item在类库中都对应一个View组件，组件属性与下面的item属性一样，可以在布局文件中使用，不像Item一样受限于ListView。

XML示例：

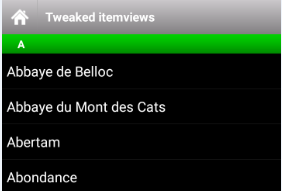
|  |
| --- |
| <description-item  greendroid:text=*"renxiuhu"* />  <separator-item  greendroid:text=*"SeparatorItem"* />  <thumbnail-item  greendroid:enabled=*"false"*  greendroid:text=*"ThumbnailItem (disabled)"*  greendroid:subtitle=*"renxiuhu"*  greendroid:thumbnail=*"@drawable/ic\_gdcatalog"* />  <drawable-item  greendroid:enabled=*"false"*  greendroid:text=*"DrawableItem (disabled)"*  greendroid:drawable=*"@drawable/ic\_gdcatalog"* />  <progress-item  greendroid:isInProgress=*"true"*  greendroid:text=*"ProgressItem"* />  <text-item  greendroid:text=*"TextItem"* />  <subtitle-item  greendroid:text=*"SubtitleItem"*  greendroid:subtitle=*"renxiuhu"* /> |

Java代码解析：

|  |
| --- |
| adapter = ItemAdapter.*createFromXml*(**this**, R.xml.*items*);  setListAdapter(adapter); |

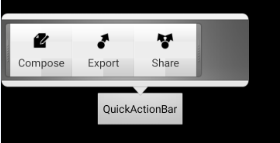
## 2.2 通讯录样式与滑动

直接参考demo中TweakedItemViewActivity的实现，可以在demo的基础上改变显示的内容和对应的操作。效果如下：



## 2.3 快捷弹出窗口

实现的效果如下所示：



* **QuickActionBar**：

——第一个图是QuickActionBar，其父类是QuickActionWidget(addQuickAction)，其可以将多个快捷操作(QuickAction)显示在一行中，可以左右滑动选择需要的操作，默认支持多个操作和滑动。

|  |
| --- |
| //实例化QuickActionWidget，添加QuickAction  QuickActionWidget mBar= **new** QuickActionBar(**this**);  //添加QuickAction，参数为context，要显示的图标，操作文本  mBar.addQuickAction(**new** QuickAction(**this**, R.drawable.*gd\_action\_bar\_add*, "添加"));  mBar.addQuickAction(**new** QuickAction(**this**, R.drawable.*gd\_action\_bar\_add*, R.string.*gd\_share*));  //设置监听器  mBar.setOnQuickActionClickListener(mActionListener);  mBar.show(v);//绑定在要弹出quickActionBar的组件上显示 |
| **private** OnQuickActionClickListener mActionListener = **new** OnQuickActionClickListener() {  **public** **void** onQuickActionClicked(QuickActionWidget widget, **int** position) {  Toast.*makeText*(QuickActionActivity.**this**, "Item " + position + " clicked", Toast.*LENGTH\_SHORT*).show();  }  }; |

* **QuickActionGrid**：

——第二个展示的就是QuickActionGrid，与QuickActionBar在代码操作上是一样的，不同的是Grid会将多个QuickAction分为多行显示。

|  |
| --- |
| QuickActionWidget mGrid = **new** QuickActionGrid(**this**);  mGrid.addQuickAction(**new** QuickAction(**this**, R.drawable.*gd\_action\_bar\_add*, "添加"));  mGrid.addQuickAction(**new** QuickAction(**this**, R.drawable.*gd\_action\_bar\_add*, R.string.*gd\_share*));  mGrid.setOnQuickActionClickListener(mActionListener);  mGrid.show(v); |

## 2.4 实现Viewflow效果分屏

使用greenDroid中的PagedView和PageIndicator可以实现和Viewflow一样的效果，只是比Viewflow要稍微复杂一点，PagedView是分屏组件，PageIndicator是圆点指示器，PageIndicator可以处理点击操作（viewflow也可以）。因为在greenDroid中PagedView和PageIndicator没有必然的联系，所以一个PagedView可以对应多个PageIndicator，每个PageIndicator的显示可以不一样。

布局文件：

|  |
| --- |
| <greendroid.widget.PagedView  android:id=*"@+id/paged\_view"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"fill\_parent"* />  <greendroid.widget.PageIndicator  android:id=*"@+id/page\_indicator\_other"*  android:layout\_width=*"wrap\_content"*  android:layout\_height=*"wrap\_content"*  android:layout\_gravity=*"top|center\_horizontal"*  //不设置dotType默认圆点全部显示，当前页面高亮；设置为multiple则只会显示圆点个数  greendroid:dotType=*"multiple"*  android:paddingTop=*"10dp"* /> |

Activity代码：

|  |
| --- |
| 1，PagedView的操作  //设置页面滑动的监听器  pagedView.setOnPageChangeListener **new** OnPagedViewChangeListener() {  @Override  **public** **void** onStopTracking(PagedView pagedView) {  }  @Override  **public** **void** onStartTracking(PagedView pagedView) {  }  @Override  **public** **void** onPageChanged(PagedView pagedView, **int** previousPage, **int** newPage) {  setActivePage(newPage);  }  };  //设置要显示的adapter  pagedView.setAdapter(**new** PhotoSwipeAdapter());  //手动设置pageview变化  pagedView.smoothScrollToNext();//向后滑动  pagedView.smoothScrollToPrevious();//向前滑动 |
| 2，PageIndicator的操作：  //设置有多少个指示圆点  mPageIndicatorOther.setDotCount(*PAGE\_COUNT*);  //设置当前点亮的指示圆点，类型为multiple时则是圆点的个数  mPageIndicatorOther.setActiveDot(page);  //为指示器设置监听器  mPageIndicatorNext.setOnClickListener(**new** OnClickListener() {  **public** **void** onClick(View v) {  pagedView.smoothScrollToNext();  }  }); |

# 3, Spring Android

Spring Android 是 Spring 框架的扩展，用于简化 Android 本地应用程序的开发。通过内嵌的RestTemplate组件，SFA项目可为您的项目提供便利的REST客户端开发方式，这种用法很像spring提供的jdbc模板类，能有效的简化客户端的开发工作量。同时，SFA也将一系列健壮的、基于开放授权的Spring社交功能集成到您的应用中，这将为您提供实现流行的社交网站类似Twitter、非死不可等的授权连接提供便利之法。

## 3.1 RestTemplate 组件

RestTemplate类是SFA RestTemplate组件包的核心类。他在概念上类似Spring框架其他同级项目中提供的模板类。RestTemlpate的行为表现取决于他提供的那些回调方法，同时通过配置合适的HttpMessageConverter 类，用户可以高效的将对象转换成HTTP请求中的信息或从响应消息中将信息转换回对应的对象。

<http://blog.csdn.net/yoara/article/details/37963497>

RestTemplate对以下功能提供了支持：

HttpClient，Gzip传输压缩，Json解析，xml解析，Rss和Atom摘要文件支持。

**构造函数**：

|  |
| --- |
| * RestTemplate(); * RestTemplate(boolean includeDefaultConverters); * RestTemplate(ClientHttpRequestFactory requestFactory); * RestTemplate(boolean includeDefaultConverters, ClientHttpRequestFactory requestFactory); |
| 默认的构造方法不包含消息的转换器，因此您必须自己添加消息转换器  RestTemplate restTemplate = new RestTemplate();  restTemplate.getMessageConverters().add(new StringHttpMessageConverter()); |

## 3.2 消息转换器

对象通过getForObject(), getForEntity(), postForLocation(), postForEntity(), postForObject() and put() 发送或者返回消息时都是通过HttpMessageConverter 的具体实现，从而转换成Http请求或者从Http应答中转换。

基于性能的选择考虑，默认的RestTemplate无参构造方法不包含任何消息转换器。但是，如果你调用了布尔版本的构造方法并传递true，则构造器会为一些主要的mime类型增加转换器。当然您自己也可以编写自定义的转换器，并通过messageConverters属性添加。  
    RestTemplate生成的转换器包:ByteArrayHttpMessageConverter, StringHttpMessageConverter, 和ResourceHttpMessageConverter。如果Android的版本在2.2或以上，那么XmlAwareFormHttpMessageConverter 和SourceHttpMessageConverter也会被添加。

**SFA自带消息转换器**：

* ByteArrayHttpMessageConverter：

该转换器可以从HTTP请求或应答中读写字节数组，默认情况下，该转换器支持所有的media types (\*/\*)，同时写数据会以Content-Type为application/octet-stream（任意二进制数据）的形式

* FormHttpMessageConverter：

能处理格式化的数据。支持用application/x-www-form-urlencoded(格式化编码数据)和multipart/form-data（多组件格式数据）的读写转换，格式化的数据通常用一个MultiValueMap<String, String>数据结构来读写

* XmlAwareFormHttpMessageConverter：

是FormHttpMessageConverter的扩展，通过SourceHttpMessageConverter增加了对XML转换的支持

* ResourceHttpMessageConverter：

能处理资源。支持对所有类型的读操作资源支持application/octet-stream

* SourceHttpMessageConverter：

可以将请求或应答数据转换成javax.xml.transform.Source类型，只支持DOMSource, SAXSource, 和StreamSource三种类型。默认情况下支持text/xml和 application/xml两种XML可扩展标记语言MIME类型

* StringHttpMessageConverter：

可以将请求或应答数据转换成String类型，支持所有的text media类型(text/\*)，支持Content-Type为text/plain（原文数据）写数据

* MappingJackson2HttpMessageConverter：

基于Jackson框架提供对请求与应答数据的Json-Obejct转换。对于Json的映射配置可以Jackson提供的注解支持。当串行化/反串行化需要时也支持切面式的控制管理，默认支持application/json类型的读写

* GsonHttpMessageConverter：

基本上和MappingJackson2HttpMessageConverter的功能一致，也支持注解和切面式。注意该转换器和MappingJackson2HttpMessageConverter转换器都支持application/json类型的数据转换，因此最好只添加一款转换器，因为RestTemplate 会优先选择他最早匹配到的转换器进行数据转换，所以两款转换器都添加会导致不可预料的结果

* SyndFeedHttpMessageConverter：

基于Android ROME Feed Reader包用于读写RSS和Atom feeds的转换器。数据将被转换成com.google.code.rome.android.repackaged和com.sun.syndication.feed.synd.SyndFeed类型。默认情况下支持application/rss+xml 和application/atom+xml两种MIME类型。SyndFeedHttpMessageConverter是RssChannelHttpMessageConverter和AtomFeedHttpMessageConverter的集合体，如果您只需要其中之一，您可以使用任意一个转换器替换SyndFeedHttpMessageConverter

## 3.3 应用示例

参照知识手册

# 4，