宝贵建议请发送至：wangzhenyang@itcast.cn

黑马程序员 itheima.com 上海 -编程，始于黑马

Android 课程同步笔记

Beta 0.02 版

By 阳哥

1、 项目说明（★）……2
2、 项目入口 MainActivity（★★）……2
2.1 MainActivity 代码清单……2
2.2 Fragment 生命周期……5
3、 系统主界面 HomeFragment（★★★★）……7
3.1 Fragment 基类 BaseFragment……8
3.2 HomeFragment……9
3.3 NewsCenterPager……15
3.3.1 NewsCenterPager……15
3.3.2BasePager……18
3.3.3 NewCenter……20
3.3.4 GsonUtil……21
3.3.5 NewPager……21
3.4 MyViewPager……41
3.4.1 MyViewPager……41
3.4.2 LazyViewPager……42
4、 菜单 MenuFragment（★★★）……43
4.1 MenuFragment 布局……43
4.2 MenuFragment 代码……45
5、 项目下载（0）……47

Android 智慧北京-03 核心业务逻辑的实现

1、项目说明（★）

写了这么多篇文档，这一篇是本人感觉目前为止最不好写的一篇！我思考了几天最终决定让文档以接近 Eclipse 中项目结构的形式来展示代码的实现。在代码中加入近乎比代码本身还多的注释。

在上一篇文章中，我们把智慧北京的整体框架给搭建起来了，在这一篇我将把智慧北京的所有核心代码展示出来，由于这个项目不像手机卫士那样，按照一个功能模块一个功能模块的讲解，而是先搭建整体框架，然后在框架中填充代码实现业务逻辑。

这个项目的业务逻辑稍微复杂一些，很难单独将其中的某个模块拿出来进行讲解，因此我就以类为基础进行讲解，在类的方法上和代码中加上详细的注释。

2、项目入口 MainActivity（★★）

2.1 MainActivity 代码清单

在 MainActivity 中生成了两个 Fragment，一个是代表主页面的 HomeFragment，另外一个是代表菜单的 MenuFragment。代码中的注释已经超过代码量，敬请查看注释。

```
public class MainActivity extends SlidingFragmentActivity {

    private SlidingMenu slidingMenu;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置改 Activity 的样式
        requestWindowFeature(Window.FEATURE_NO_TITLE);
```

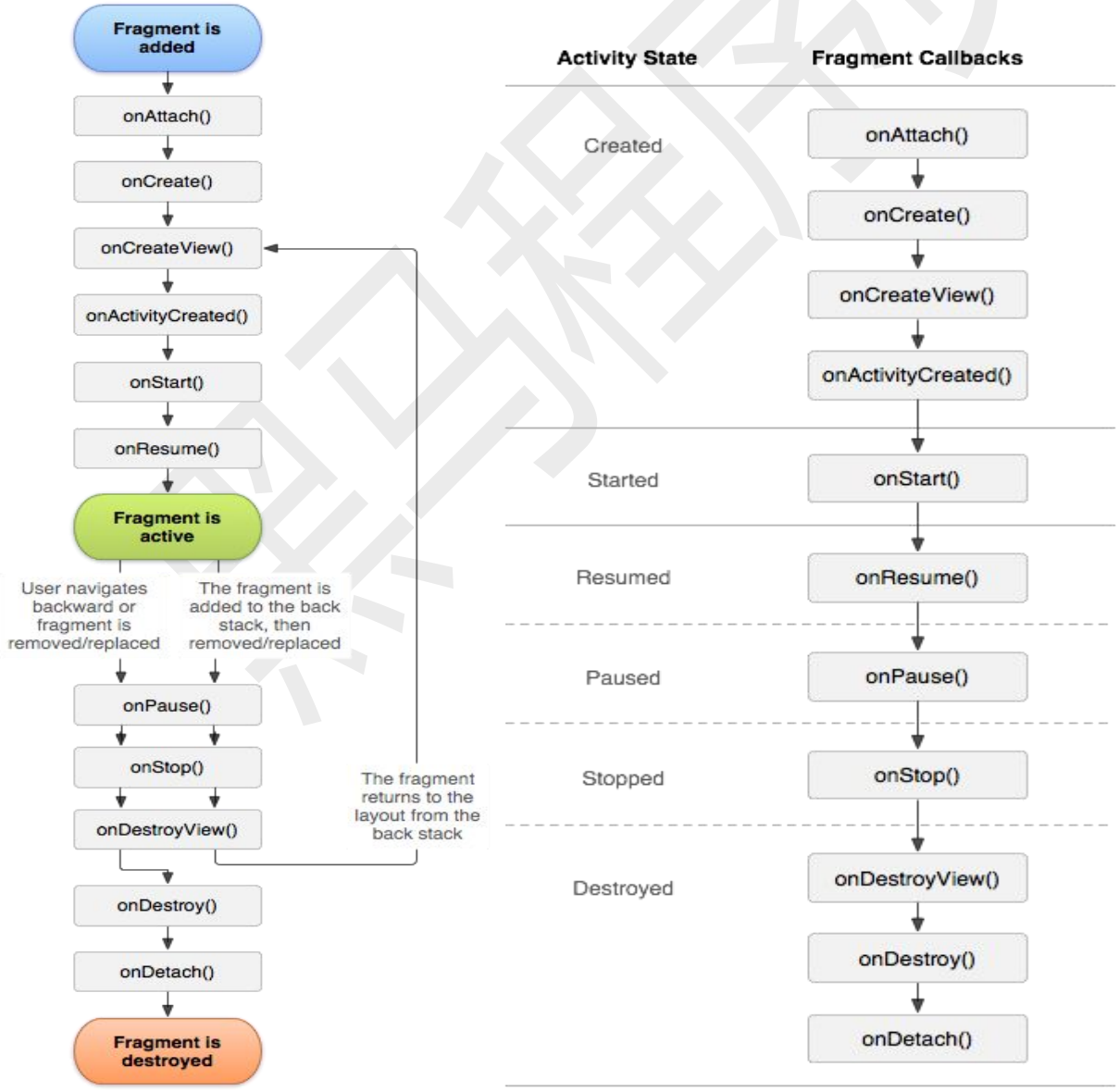
```
        setContentView(R.layout.content);
        /*
         * 给 SlidingMenu 设定布局文件，该布局文件就是一个 FrameLayout，等待着被替换，
FrameLayout 这个控件生来命就苦，
         * 基本用到最多的 就是作为 Fragment 的布局替身，或者其他 View 来替换，一直扮演着占位
的角色
         */
        setBehindContentView(R.layout.menu_frame);
        // 因为我们当前 Activity 已经继承了 SlidingFragmentActivity，所有我们可以直接获取
SlidingMenu 对象
        slidingMenu = getSlidingMenu();
        /*
         * 给 SlidingMenu 设置模式，这里我们采用左侧的模式 还有右侧，左右同时存在等模式
         * 所谓的左侧模式其实就是菜单居于当前 Activity 的左侧
         */
        slidingMenu.setMode(SlidingMenu.LEFT);
        /*
         * 设置 SlidingMenu 的宽度 这里是通过 dimens.xml 文件中的值来设定，其这样的做法应该
是为了适配不同分辨率的手机考虑的
         * dimens.xml 中的对应值为： <dimen name="slidingmenu_offset">180dp</dimen>
         */
        slidingMenu.setBehindOffsetRes(R.dimen.slidingmenu_offset);
        /*
         * 给 SlidingMenu 设置一个阴影背景，所谓的阴影背景其实我们可以理解为菜单边缘处的颜
色，一般都是渐变的色彩， 这里设置背景 也是需要在 xml 文件中设置
         * shadow.xml 文件清单如下，该文件是渐变色
         * <?xml version="1.0" encoding="utf-8"?>
         * <shape xmlns:android="http://schemas.android.com/apk/res/android" >
         * <gradient android:endColor="#5A000000"
         * android:centerColor="#2D000000" android:startColor="#00000000" />
         * </shape>
         */
        slidingMenu.setShadowDrawable(R.drawable.shadow);
        /*
         * 给菜单设置阴影的宽度，其值如下
         *
         * <dimen name="shadow_width">5dp</dimen>
         */
        slidingMenu.setShadowWidthRes(R.dimen.shadow_width);
        /*
         * 给 SlidingMenu 设置触摸模式
         * 因为 SlidingMenu 默认是隐藏（上面已经设置为隐藏在左侧）的，如果想滑动出来，我们
需要在当前界面在哪个位置滑动呢？
```

```
        * SlidingMenu 给我们提供了 2 中模式，或者说 3 中如下：
        * SlidingMenu.TOUCHMODE_MARGIN
        * SlidingMenu.TOUCHMODE_FULLSCREEN
        * SlidingMenu.TOUCHMODE_NONE
        * 如果英语还可以，其实每一个类名，每一个方法名，每一个参数名都是最好的注释
        * 第一种是边缘，也就是说我们从左侧边缘往右滑动手指才可以滑出菜单
        * 第二种是全屏幕，在屏幕的任何位置往右滑动都可以滑出菜单
        * 第三种是不能滑出菜单，就是不让你把菜单滑出
        */
       slidingMenu.setTouchModeAbove(SlidingMenu.TOUCHMODE_FULLSCREEN);
       /*
        * 创建菜单 Fragment
        * 将 SlidingMenu 所有的布局替换成 MenuFragment
        */
       MenuFragment menuFragment = new MenuFragment();
       getSupportFragmentManager().beginTransaction().replace(R.id.menu,
menuFragment, "MENU").commit();
       /*
        * 创建住 Fragment，HomeFragment
        * 该 Fragment 用的是 MainActivity 的布局文件，将当前 Activity 替换成该 Fragment
        */
       HomeFragment homeFragment = new HomeFragment();
       getSupportFragmentManager().beginTransaction().replace(R.id.content_frame,
homeFragment, "HOME").commit();

   }
   /*
    * 对外提供的方法，这样外部其他类只要拥有 content（也就是当前 Activity）就可以获取该类
中的 Fragment
    * 这两个方法对外暴漏了 MainActivity 中的 2 个 Fragment
    * 查询 Fragment 时根据 Tag 来查找的，如果我们把两个 Fragment 当做成员变量也是可以的。
    * 你可能会问，为何能拿到 Activity，就能拿到这个 Activity 种的 Fragment 呢？
    * 这是因为 Fragment 的生命周期本身就是绑定在 Activity 的，也就说 Fragment 本省就是
Activity 的附属
    * 这一点从 Fragment 生命周的 onAttach（绑定）onDetach（解绑也能看得出）
    */
   public MenuFragment switchMenuFragment() {
       return (MenuFragment) getSupportFragmentManager().findFragmentByTag("MENU");
   }

   public HomeFragment switchHomeFragment() {
       return (HomeFragment) getSupportFragmentManager().findFragmentByTag("HOME");
   }}
```

Tips：在上面代码的注释中我提到了为何在 Activity 中能获取其中的 Activity。我说这是因为 Fragment 是绑定在 Activity 中的。那么 Fragment 的生命周期是什么呢？

2.2 Fragment 生命周期

Fragment 的生命周期,我在 Android 基础文档中已经详细介绍过，能够看到这里的也应该是基础和耐心都比较 OK 的吧，因此为了节约时间同时帮大家复习 Fragment 生命周期知识，我把 Fragment 生命周期的官方流程图贴出来。考虑到 Fragment 和 Activity 千丝万缕的关系，因此也贴出 Activity 和 Fragment 生命周期的对比图。这个内容也是面试的“重灾区”。



从上面可以看出来 Fragment 的生命周期要比 Activity 多，我们可以这么理解。其实我们所谓的“生命周期”只是用系统的回调函数来代替罢了。上面的那些方法都是 Activity 或者 Fragment 状态改变时的回调过程，是一个被动的过程，而不是说我们自己调用了上面的任何方法就改变了其生命周期。

Fragment 生命周期场景演示：

- 切换到该 Fragment

15-3 16:26:35.095: D/AppListFragment(7649): onAttach

15-3 16:26:35.095: D/AppListFragment(7649): onCreate

15-3 16:26:35.095: D/AppListFragment(7649): onCreateView

15-3 16:26:35.100: D/AppListFragment(7649): onActivityCreated

15-3 16:26:35.120: D/AppListFragment(7649): onStart

15-3 16:26:35.120: D/AppListFragment(7649): onResume

- 屏幕灭掉

15-3 16:27:35.185: D/AppListFragment(7649): onPause

15-3 16:27:35.205: D/AppListFragment(7649): onSaveInstanceState

15-3 16:27:35.205: D/AppListFragment(7649): onStop

- 屏幕解锁

15-3 16:33:13.240: D/AppListFragment(7649): onStart

15-3 16:33:13.275: D/AppListFragment(7649): onResume

- 切换到其他 Fragment

15-3 16:33:33.655: D/AppListFragment(7649): onPause

15-3 16:33:33.655: D/AppListFragment(7649): onStop

15-3 16:33:33.660: D/AppListFragment(7649): onDestroyView

- 切换回本身的 Fragment

15-3 16:33:55.820: D/AppListFragment(7649): onCreateView

15-3 16:33:55.825: D/AppListFragment(7649): onActivityCreated

15-3 16:33:55.825: D/AppListFragment(7649): onStart

15-3 16:33:55.825: D/AppListFragment(7649): onResume

- 回到桌面

15-3 16:34:26.590: D/AppListFragment(7649): onPause

15-3 16:34:26.880: D/AppListFragment(7649): onSaveInstanceState

15-3 16:34:26.880: D/AppListFragment(7649): onStop

- 回到应用

15-3 16:36:51.940: D/AppListFragment(7649): onStart

15-3 16:36:51.940: D/AppListFragment(7649): onResume

- 退出应用

15-3 16:37:03.020: D/AppListFragment(7649): onPause

15-3 16:37:03.155: D/AppListFragment(7649): onStop

15-3 16:37:03.155: D/AppListFragment(7649): onDestroyView

15-3 16:37:03.165: D/AppListFragment(7649): onDestroy

15-3 16:37:03.165: D/AppListFragment(7649): onDetach

3、系统主界面 HomeFragment（★★★★）

在 MainActivity 中我们用到了两个 Fragment，这是已知的使用量。在真实项目的不断更新过程中，可能有更多的 Fragment 需要添加到项目中（尽管我们演示的没有），因此对一些共性的类就行向上抽取不仅可以提高我们的代

码重用度也增加了代码的可移植性。

基于上面的考虑，我们抽取了 BaseFragment，该类作为该项目中其他 Fragment 的父类。

3.1 Fragment 基类 BaseFragment

在 BaseFragment 类中，我们将 Context、SlidingMenu、View 都作为我们的成员变量并且声明为 public 的了，其实我们不声明为 public 对外提供一个 getter 方法也是可以的。那么为什么将这些对象拿到呢，因为在子 Fragment 中我们 HomeFragment 需要操作 MenuFragment 对象中的内容，MenuFragment 也需要操作 HomeFragment 对象中的内容，毕竟菜单肯定是和内容关联起来的，不同的内容对应不同的菜单，不同的菜单反过来也对应不同的内容。

```
/*
 * 项目中其他 Fragment 的基类，包含了 inteView 和 initData 两个抽象方法，因此该类也是抽象的
 */
public abstract class BaseFragment extends Fragment {
    public View view;
    public Context context;
    public SlidingMenu slidingMenu;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /*
         * 获取 Context 为何是 getActivity 呢？
         * 很简单因为 Activity 是 Context 的子类呗
         * 为什么在 Fragment 中可以获取到 Activity（Context）呢？
         * 以还是那句话因为 Fragment 是绑定在 Activity 中的，因此 Fragment 提供了
         * getActivity 方法，可以让我们方便的获取到当前 Fragment 绑定的 Activity 对象
         */
        context = getActivity();
        /*
         * 获取 SlidingMenu 对象
         * 为什么在 context 中能获取到 SlidingMenu 对象呢，因为我们的 MainActivity
         * 是继承了 SlidingFragmentActivity 类的。
         */
        slidingMenu = ((MainActivity)context).getSlidingMenu();
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
```

```
            Bundle savedInstanceState) {
        //将初始化 View 的方法房子抽象函数中，具体创建什么样的 View 交给子类来实现
        view = initView(inflater);
        return view;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        //初始化数据也方法也声明为抽象的，让子类来实现
        initData(savedInstanceState);
        super.onActivityCreated(savedInstanceState);
    }
    //声明两个抽象方法
    public abstract void initData(Bundle savedInstanceState);
    public abstract View initView(LayoutInflater inflater);
}
```

3.2 HomeFragment

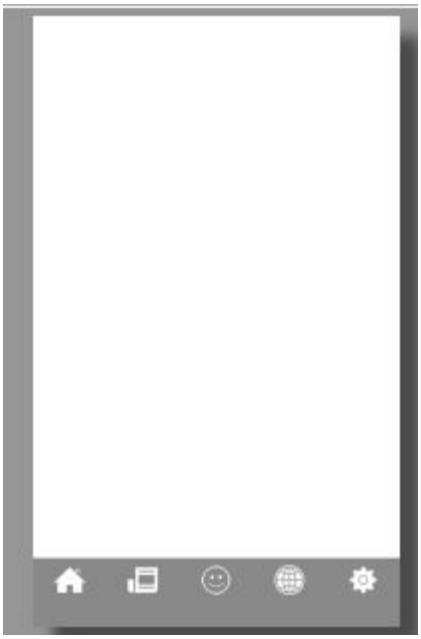
该类继承自 BaseFragment。该类主要用于显示内容的主界面，这里有必须要将该类用到的布局清单和效果图展示给大家。

- frag_home.xml 清单如下：

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

    <!-- 自定义的 ViewPager 控件。我必须说明的是这里的 ViewPager 可不是图上那个不停滚动的新闻图片，
    而是我们他最下端的 tab，切换 tab 的时候就是切换这个 ViewPager -->

    <com.itheima.zhbj_heima.view.MyViewPager
        android:id="@+id/layout_content"
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_weight="1.0" />
```

```
    <!-- 在 RadoGroup 中放置了多个 RadioButton，点击 RadioButton 触发的效果就是切换我们上面自定义的 ViewPager-->
    <RadioGroup
        android:id="@+id/main_radio"
        android:layout_width="fill_parent"
        android:layout_height="60dp"
        android:layout_gravity="bottom"
        android:background="@drawable/bottom_tab_bg"
        android:gravity="center_vertical"
        android:orientation="horizontal"
        android:paddingTop="2dp" >

        <RadioButton
            android:id="@+id/rb_function"
            style="@style/main_tab_bottom"
            android:drawableTop="@drawable/icon_function"
            android:text="@string/tab_function" />

        <RadioButton
            android:id="@+id/rb_news_center"
            style="@style/main_tab_bottom"
            android:drawableTop="@drawable/icon_newscenter"
            android:text="@string/tab_news_center" />

        <RadioButton
            android:id="@+id/rb_smart_service"
            style="@style/main_tab_bottom"
            android:drawableTop="@drawable/icon_smartservice"
            android:text="@string/tab_smart_service" />

        <RadioButton
            android:id="@+id/rb_gov_affairs"
            style="@style/main_tab_bottom"
            android:drawableTop="@drawable/icon_govaffairs"
            android:text="@string/tab_gov_affairs" />
        <RadioButton
            android:id="@+id/rb_setting"
            style="@style/main_tab_bottom"
            android:drawableTop="@drawable/icon_setting"
            android:text="@string/tab_setting" />
    </RadioGroup>
</LinearLayout>
```

Tips：在上面的布局文件中我们发现其实只有两个大控件，一个是自定义的 ViewPager 类，MyViewPager，另外一个是 RadioGroup，RadioGroup 里面包含了 N 个 RadioButton。我们自定义的 ViewPager 是不自动切换的，只有当点击 RadioButton 的情况下才让其换页。该布局预览图如下：



HomeFragment 代码

```
/**
 * 主界面，这个主界面是一个 ViewPager 和 RadioGroup 组成，点击 RadioGroup 中的 RadioButton 可以切换 ViewPager
 */
public class HomeFragment extends BaseFragment {
    public static final String   tag= "HomeFragment";
    @ViewInject(R.id.layout_content)
    private MyViewPager      viewPagerlayout_content;
    @ViewInject(R.id.main_radio)
    private RadioGroup       radioGroup;
    private List<BasePager>      pagersList;

    @Override
    public View initView(LayoutInflater inflater) {
        /*
         * 初始化视图，填充布局文件
         */
        view = inflater.inflate(R.layout.frag_home, null);
        /*
```

```
        * 这里使用 xUtils 工具中的 ViewUtils
        * 通过将 view 注入给 ViewUti，然后我们就能通过注解的形式获取 view 中的子控件，原理其实很简单，
        * 首先通过反射获取当前类（this）,然后获取该类的属性（暴力反射）上的注解，然后从 view 中根据属性上的注解 id 拿到子控件，
        * 然后反射 findViewByID（）方法，将生成的对象暴力反射给该类的属性，OK 了，我们就可以直接使用子控件了，就这么简单。
        */
       ViewUtils.inject(this, view);
       return view;
   }
   /*
    * 覆写父类的方法，在该方法中初始化数据
    * 初始化 RadioGroup
    * 初始化 ViewPager（MyViewPager）
    * 给 ViewPager 设定 Pager，BasePager
    */
   @Override
   public void initData(Bundle savedInstanceState) {
       //RadioGroup 总得有个默认选中项吧，那好，我们就把第一个子 RadioButton，首页设定为默认选中项
       radioGroup.check(R.id.rb_function);
       //创建一个 ArrayList，用于存储我们 5 个 tab 对应的 pager，这个 list 最后还是会作为 myViewPager 的数据来源
       pagersList = new ArrayList<BasePager>();
       /*
        * 给 list 添加 pager
        * 这些 Pager，每一个都是对应 tab 的一个选项
        * 他们继承自 BasePager
        */
       pagersList.add(new FuctionPager(getActivity()));//首页
       pagersList.add(new NewsCenterPager(getActivity()));//新闻中心
       pagersList.add(new SmartServicePager(getActivity()));//智慧服务
       pagersList.add(new GovAffairsPager(getActivity()));//政务指南
       pagersList.add(new SettingPager(getActivity()));//设置
       /*
        * 给 MyViewPager 的对象 viewPagerlayout_content 设置适配器
        * ViewPager 的使用在自定义控件课程中学过，因此不多解释这句代码
        */
       viewPagerlayout_content.setAdapter(new MyAdapter());
       /*
        * 给 MyViewPager 对象 viewPagerlayout_content 设置页面更改监听事件，就是当 ViewPager 切换
```

```
        * 页面的的时候调用的方法
        * 在该方法中我会做哪些操作呢？
        * 大家想呢，页面切换了，是不是要给当前切换到页面初始化数据呢？对的，就是给切换到的
页面初始化数据
        */
       viewPagerlayout_content.setOnPageChangeListener(new OnPageChangeListener() {
          @Override
          public void onPageSelected(int position) {
             //获取当前页对应的 pager，从 list 中拿出来然后进行初始化数据的操作
             BasePager basePager = pagersList.get(position);
             basePager.initData();
          }

          @Override
          public void onPageScrolled(int position, float positionOffset, int
positionOffsetPixels) {
             // TODO Auto-generated method stub
          }

          @Override
          public void onPageScrollStateChanged(int state) {
             // TODO Auto-generated method stub
          }
       });
       /*
        * 给 RadioGroup 设置选中监听事件
        * 当某一个 RadioButton 被选中时回调该方法
        */
       radioGroup.setOnCheckedChangeListener(new OnCheckedChangeListener() {
          @Override
          public void onCheckedChanged(RadioGroup group, int checkedId) {
             /* 判断当前选中的是哪个 RadioButton
              * 然后给 MyViewPager 对象设置当前的 Pager 页
              */
             switch (checkedId) {
             case R.id.rb_function:
                viewPagerlayout_content.setCurrentItem(0);
                break;
             case R.id.rb_news_center:
                viewPagerlayout_content.setCurrentItem(1);
                break;
             case R.id.rb_smart_service:
                viewPagerlayout_content.setCurrentItem(2);
```

```
                break;
            case R.id.rb_gov_affairs:
                viewPagerlayout_content.setCurrentItem(3);
                break;
            case R.id.rb_setting:
                viewPagerlayout_content.setCurrentItem(4);
                break;
            }
        }
    });
    //因为我们的 RadioGroup 默认把第一页当做默认页，因此我们给第一页初始化数据
    BasePager basePager = pagersList.get(0);
    basePager.initData();
}
/*
 * 自定义 ViewPager 的适配器
 */
class MyAdapter extends PagerAdapter {
    @Override
    public int getCount() {
        return pagersList.size();
    }

    @Override
    public boolean isViewFromObject(View arg0, Object arg1) {
        return arg0 == arg1;
    }
    /*
     * 初始化 ViewPager 要展示的内容
     * ViewPager 要展示的每一个 Pager 都是我们 initData 方法中放到集合中的
     * 这里需要从集合中拿出来，然后获取其要展示的 View 对象
     */
    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        ((MyViewPager)
container).addView(pagersList.get(position).getRootView());
        return pagersList.get(position).getRootView();
    }

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

```
    }

    public NewsCenterPager switchNewCenterPager() {
        return (NewsCenterPager) pagersList.get(1);
    }
}
```

3.3 NewsCenterPager

在 HomeFragment 中的 MyViewPager 每一页都是一个 Pager，我们的首页、新闻中心、智慧服务、政务指南、设置等是 MyViewPager 的 5 个 Pager，因为我们项目主要完成新闻中心，因此这里给出新闻中心对应的 NewsCenterPager 类的实现。

3.3.1 NewsCenterPager

```
public class NewsCenterPager extends BasePager {

    protected static final String tag = "NewsCenterPager";
    //我们将从网络上加载来的数据封装在 NewCenter 这个 Bean 中
    private NewCenter newCenter;
    //通过注解的形式初始化布局文件中的帧布局
    @ViewInject(R.id.news_center_fl)
    public FrameLayout news_center_fl;
    /*
     * 这里为什么要用一个集合来存储这么多 pager 呢，因为我们当前 NewsCenterPager 其实就是一个 FameLayout，从布局文件中也知道。并不提供真正的内容
     * 真正的是左侧菜单中具体选项来决定的。我们可以将我们项目的底部作为一级菜单，然后左侧的 SlidingMenu 作为二级菜单
     */
    private List<BasePager> pagers;
    /*
     * 跟 Pager 对应的标题，如果问为何不讲替他作为 BasePager 的属性放进去呢，其实这样设计也行，为什么不这样设计是因为我们将 title 的布局已经抽取出来了
     * 其他 tab 中的选项也可以用跟这个布局文件，从当前类用到的布局文件中引入的布局我们也能看得出其设计用意
     */
    private List<String> titleList;
```

```
    public NewsCenterPager(Context context) {
        super(context);
    }
    //初始化布局文件
    @Override
    public View initView() {
        view = View.inflate(context, R.layout.news_center_frame, null);
        //提供注解支持
        ViewUtils.inject(this, view);
        //初始化标题
        initTitleBar();
        return view;
    }

    @Override
    public void initData() {
        /*
         * 我在写智慧北京的第三篇文章时，我发现在前面的文章中犯了一个严重错误
         * 我把 tomcat 中用到的文件的地址改成 localhost 了，其实应该改成 10.0.2.2，为啥大家
都懂的
         * http://10.0.2.2:8080/qbc/categories.json
         * 这里是从 sp 中获取新闻数据，如果有则直接使用，如果没有则从网络中加载
         * sp 的 key 就是要请求的地址，如果数据比较多我不建议大家在以后的开发中将数据存储在
sp 中，因为 sp 是基于 xml 的
         * xml 的存储和解析本身效率就不高
         */
        String result = SharedPreferencesUtil.getStringData(context,
HMApi.NEWS_CENTER_CATEGORIES, "");
        if(!TextUtils.isEmpty(result)){
            //有数据,解析
            processData(result);
        }else {
            //没数据请求数据
            getNewCenterData();
        }
    }

    private void getNewCenterData() {
        /*
         * getData 方法定义在父类 BasePager 里面
         * 用的是 xUtils 中 HttpUtils 工具类
         */
        getdata(HttpMethod.GET, HMApi.NEWS_CENTER_CATEGORIES, null, new
```

```
RequestCallBack<String>() {
          @Override
          public void onSuccess(ResponseInfo<String> responseInfo) {
              //数据请求成功后保存到 xml 中
              SharedPreferencesUtil.saveStringData(context,
HMApi.NEWS_CENTER_CATEGORIES, responseInfo.result);
              //然后处理数据
              processData(responseInfo.result);
          }

          @Override
          public void onFailure(HttpException error, String msg) {
              //当请求网络失败时的回调函数
          }
      });
   }

   //解析数据操作
   private void processData(String result) {
       /*
        * 使用开源项目 Gson 对 json 数据进行解析，我们 GsonUtil 是自己创建的一个工具类
        * 该工具的一大用点，就是可以直接将 json 字符串转化为 JavaBean
        * 这里把新闻中心数据转化为 NewCenter Bean.
        * 我们定义的 JavaBean 中属性必须跟 json 中的可以保持一致才可以转化
        */
       newCenter = GsonUtil.jsonToBean(result, NewCenter.class);
       titleList = new ArrayList<String>();
       for(int i=0; i<newCenter.data.size();i++){
           //获取所有的标题
           titleList.add(newCenter.data.get(i).title);
       }
       //当前 titleList 是封装了左侧侧拉条目的数据，需要传递给 MenuFragment，获取
MenuFragment 对应对象
       ((MainActivity)context).switchMenuFragment().initMenu(titleList);
       /*
        * 创建 4 个 Pager 对象
        * 大家可能已经晕了，不知道下面的四个 Pager 是哪些，这些 Pager 是当我们点击新闻中心
这个 RadioButton 的时候
        * 在左侧会初始化菜单栏，菜单栏里有四个选项，分别是新闻 、话题、图片、互动，默认是
新闻
        * 选中新闻的时候，那么在主界面就显示出 NewPager
        */
       pagers = new ArrayList<BasePager>();
```

```
        pagers.add(new NewPager(context,newCenter.data.get(0)));
        pagers.add(new TopicPager(context,newCenter.data.get(1)));
        pagers.add(new PicPager(context,newCenter.data.get(2)));
        pagers.add(new IntPager(context,newCenter.data.get(3)));
        //默认选中新闻 我们也只做新闻
        switchPager(0);
    }

    public void switchPager(int i) {
        //左侧侧拉栏目选中指条目时，需要去加载的页面
        txt_title.setText(titleList.get(i));

        //底部帧布局
        news_center_fl.removeAllViews();
        //获取详细条目的 view 对象
        news_center_fl.addView(pagers.get(i).getRootView());
        //初始化数据
        BasePager basePager = pagers.get(i);
        basePager.initData();
    }
}
```

3.3.2BasePager

```
/*
 * 抽象类 BasePager
 */
public abstract class BasePager {
    public Context context;
    public View view;
    public SlidingMenu slidingMenu;
    //这个属性是页面的标题，这个也作为基础类是因为多有页面都有标题
    public TextView txt_title;
    //头部按钮
    private ImageButton imgbtn_text;
    private ImageButton imgbtn_right;
    private ImageButton btn_right;

    public BasePager(Context context) {
        //在构造函数中获取 Context，目的是为了获取 Context 中的 SlidingMenu 以及 Context 中
的 Fragment
        this.context = context;
```

```
        slidingMenu = ((MainActivity)context).getSlidingMenu();
        view = initView();
    }
    /*
     * 返回 initView 方法中创建的 View 对象
     * 这里这个方法显得很多余，因为我们的 View 对象已经声明为 public 了子类可以直接使用
     * 而不需调用该方法，但是从架构的思想来说通过调用期对外暴漏的方法也直接暴漏属性要好
     */
    public View getRootView(){
        return view;
    }
    /*
     * 初始化标题栏
     */
    public void initTitleBar(){
        Button btn_left = (Button) view.findViewById(R.id.btn_left);
        btn_left.setVisibility(View.GONE);
        ImageButton imgbtn_left = (ImageButton) view.findViewById(R.id.imgbtn_left);

        //前景(当前控件显示的内容)
        imgbtn_left.setImageResource(R.drawable.img_menu);
        //点击左上角的菜单图标时菜单隐藏或者显示
        imgbtn_left.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                //SlidingMenu 自动完成隐藏或者显示，如果显示状态则隐藏，相反则相反
                slidingMenu.toggle();
            }
        });
        /*
         * 将头部没有用到的控件获取到然后隐藏掉
         * 如果你问既然这些控件没有用为何要写在布局文件中，
         * 这是因为这是一个公用的标题布局，我们把没用用到的都隐藏
         * 哪个子类用到了在显示出来就行了
         */
        txt_title = (TextView) view.findViewById(R.id.txt_title);
        imgbtn_text = (ImageButton) view.findViewById(R.id.imgbtn_text);
        imgbtn_right = (ImageButton) view.findViewById(R.id.imgbtn_right);
        btn_right = (ImageButton)view.findViewById(R.id.btn_right);

        imgbtn_text.setVisibility(View.GONE);
        imgbtn_right.setVisibility(View.GONE);
        btn_right.setVisibility(View.GONE);
```

```
    }

    public abstract View initView();
    public abstract void initData();
    //调用 xUtils 中的 HttpUtils 工具加载数据
    public void getdata(HttpMethod httpMethod,String url,RequestParams
params,RequestCallBack<String> callBack) {
        HttpUtils httpUtils = new HttpUtils();
        httpUtils.send(httpMethod,url,params,callBack);
    }
}
```

3.3.3 NewCenter

NewCenter 是我们根据 Json 格式的数据封装的 JavaBean。这里面我们的所有属性都是 public 的，感觉不符合 java 的封装的特性，但是这样的做法无可厚非，因为这个对象只是作为临时数据的存储，或者说是序列化数据转化成内存对象的载体，因此怎么方便就怎么来了，而且我们的嵌套层次比较深，如果写 setter、getter 也是很麻烦。没有最好的设计思想，只有最好的应用场景，对于这种设计我们从了吧。

```
public class NewCenter {
    public List<NewCenterItem> data;
    public List<String> extend;
    public String retcode;

    public class NewCenterItem{
        public List<Children> children;
        public String id;
        public String title;
        public String type;
        public String url;
        public String url1;
        public String dayurl;
        public String excurl;
        public String weekurl;
    }

    public class Children{
        public String id;
```

```
        public String title;
        public String type;
        public String url;
    }
}
```

3.3.4 GsonUtil

```
public class GsonUtil {
    //使用 Gson 开源框架，将一个 json 格式的字符串转化为 T 对象
    public static <T> T jsonToBean(String json,Class<T> clazz){
        Gson gson = new Gson();
        return gson.fromJson(json, clazz);
    }
}
```

3.3.5 NewPager

NewPager 是我们主界面主线上的一个页面，就是新闻中心（tab 菜单）（NewsCenterPager）-新闻（左侧菜单）（NewPager）。

3.3.5.1 NewPager 布局

NewPager 布局有必要展示出来，这个布局里用到了导航条，很遗憾（或者说庆幸）的是我们的导航条用到的也是 Github 上的开源代码 ViewPagerIndicator，我们把这个类拿出来放到我们的工程中自己用了。因此这个类就不展示了，在文档的最后我会附上整个项目的下载地址，大家下载下来项目就行，项目里面啥都有。导航条如下所示：

NewPager 用到的布局文件为 frag_news.xml，清单如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#F6F6F6"
    android:orientation="vertical" >

    <LinearLayout
        android:id="@+id/ll_indicator"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
         android:background="@drawable/news_tab_bg"
        android:orientation="horizontal" >

        <!-- 当前控件可用去指定底部 viewpager 的显示,一一对应(1,1) -->
        <com.itheima.zhbj_heima.view.pagerindicator.TabPageIndicator
            android:id="@+id/indicator"
            style="@style/Theme.PageIndicatorDefaults"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:paddingBottom="2dp" />

        <ImageView
            android:id="@+id/iv_edit_cate"
            android:layout_width="30dp"
            android:layout_height="50dp"
            android:scaleType="center"
            android:src="@drawable/news_cate_arr" />
    </LinearLayout>

    <android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

</LinearLayout>
```

Tips：上面布局文件中的 TabPageIndicator 其实就是 Github 上的 ViewPagerIndicator，只不过是改了个名字。

3.3.5.2 NewPager 代码

```
public class NewPager extends BasePager {
    //自定义的 JavaBean 位于 NewCenter 类中
    private NewCenterItem newCenterItem;
    /*
     * 指示器，也叫导航栏，也叫指针，我个人倾向于叫导航栏
     * 这个导航栏在下面的代码中关联上了 ViewPager，导航栏的一个子条目就对应 ViewPager 的一个页面
     */
    @ViewInject(R.id.indicator)
    private TabPageIndicator indicator;
    @ViewInject(R.id.pager)
    private ViewPager viewPager;
    /*
     * 因为每个导航栏的子条目都对应一个 ViewPager 中的一个页面，因此我们把这些页面放到一个集合中
     */
    private List<BasePager> pagerList = new ArrayList<BasePager>();
    /*
     * 每个 pager 都得有头部信息，我们把这些头部信息也放到集合中
     */
    private List<String> titleList = new ArrayList<String>();
    //专供 ViewPager 的适配器
    private MyAdapter myAdapter;
    /*
     * 构造函数，在构造的时候第二个参数就是数据，把数据传递过来
     */
    public NewPager(Context context,NewCenterItem centerItem) {
        super(context);
        this.newCenterItem = centerItem;
    }

    @Override
    public View initView() {
        view = View.inflate(context, R.layout.frag_news, null);
        ViewUtils.inject(this,view);
        return view;
    }

    @Override
    public void initData() {
        //底部 viewpager 需要去显示一个界面，并且当前界面继承至 BasePager，显示具体内容封装在 centerItem 对应 children 节点中 url 所指向的地址
        for(int i=0;i<newCenterItem.children.size();i++){
```

```
            titleList.add(newCenterItem.children.get(i).title);
            //根据传递进来的数据创建 pager，这个 pager 就是需要展示出来的，也是我们用户最终看得见的页面
            pagerList.add(new NewItemPager(context,newCenterItem.children.get(i).url));
        }
        /*
         * 如果适配器为 null 则创建，否则 notify 更新 ViewPager 就行了
         */
        if(myAdapter == null){
            myAdapter = new MyAdapter();
            viewPager.setAdapter(myAdapter);
        }else{
            myAdapter.notifyDataSetChanged();
        }
        //指针需要和 viewpager 进行绑定，即指针指向那页，viewpager 页处在那页
        indicator.setViewPager(viewPager);
        //给导航器设定第一个子条目为选中项
        indicator.setCurrentItem(0);
        /*
         * 给 ViewPager 设置换页监听器
         * 监听器在这里主要完成 2 个工作
         * 1、当我们的导航栏已经到第一个的时候那么我们就可以抽出 SlidingMenu，导航栏没到第一个则不能抽出 SlidingMenu
         * 2、导航栏导航到的页面进行初始化操作
         */
        viewPager.setOnPageChangeListener(new OnPageChangeListener() {

            @Override
            public void onPageSelected(int arg0) {
                //只有在第一页的时候才可以拖拽出左侧侧拉栏目
                if(arg0 == 0){
                    slidingMenu.setTouchModeAbove(SlidingMenu.TOUCHMODE_FULLSCREEN);
                }else{
                    slidingMenu.setTouchModeAbove(SlidingMenu.TOUCHMODE_NONE);
                }
                indicator.setCurrentItem(arg0);
                BasePager basePager = pagerList.get(arg0);
                basePager.initData();
            }

            @Override
            public void onPageScrolled(int arg0, float arg1, int arg2) {
```

```
            // TODO Auto-generated method stub

        }

        @Override
        public void onPageScrollStateChanged(int arg0) {
            // TODO Auto-generated method stub

        }
    });
    //默认初始化导航栏第一个页面
    BasePager basePager = pagerList.get(0);
    basePager.initData();
}

class MyAdapter extends PagerAdapter{
    @Override
    public CharSequence getPageTitle(int position) {
        return titleList.get(position);
    }

    @Override
    public int getCount() {
        return titleList.size();
    }

    @Override
    public boolean isViewFromObject(View arg0, Object arg1) {
        return arg0 == arg1;
    }

    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        ((ViewPager)container).addView(pagerList.get(position).getRootView());
        return pagerList.get(position).getRootView();
    }

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

3.3.5.3 NewItemPager

在 NewPager 代码中导航栏的每一个条目都是一个 NewItemPager 对象。NewItemPager 用到的布局整体式一个 ListView，不过我们加上了下拉刷新和上拉加载数据，用到了 Github 上又一开源框架 *PullToRefreshListView*。

NewItemPager 共用到了 2 个布局文件，第一个布局文件就是我们的图片轮播图，第二个就是 ListView。

第一个布局文件 layout_roll_view.xml 清单：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="185dp"
     >
      <!-- 放置轮播图片位置 -->
       <LinearLayout
           android:id="@+id/top_news_viewpager"
           android:layout_width="fill_parent"
           android:layout_height="185dp"
           android:orientation="horizontal" />

       <LinearLayout
           android:layout_width="fill_parent"
           android:layout_height="30dp"
           android:layout_alignParentBottom="true"
           android:background="#88000000"
           android:gravity="center_vertical"
           android:orientation="horizontal" >
         <!-- 放置图片标题的位置 -->
           <TextView
               android:id="@+id/top_news_title"
               android:layout_width="0dp"
               android:layout_weight="1"
               android:layout_height="wrap_content"
               android:text="图片标题"
               android:layout_marginRight="8dp"
               android:layout_marginLeft="5dp"
               android:singleLine="true"
               android:textColor="#F6F6F6"
               />
         <!-- 放置图片中选中点的位置 -->
```

```
            <LinearLayout
                android:id="@+id/dots_ll"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginRight="5dp"
                android:gravity="center"
                android:orientation="horizontal"
                 />
        </LinearLayout>
    </RelativeLayout>
```

布局文件 frag_item_news.xml 清单如下：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/white" >
    <view
        android:id="@+id/lv_item_news"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:cacheColorHint="@android:color/white"
        android:divider="@drawable/news_list_line"
        android:fadingEdge="none"
        class="com.itheima.zhbj_heima.view.pullrefreshview.PullToRefreshListView"
        android:scrollbars="none" />
</FrameLayout>
```

NewItemePager 是我们现实页面最直接的类，也是最复杂的类。之前嵌套了那么多，这个类才是真正干活的。

NewItemPager 代码清单：

```
/**
 * NewPager 中的 viewPager 的每个子页面
 */
public class NewItemPager extends BasePager {

    private String url;
    /*
     * 轮播图用到的布局
     */
```

```
    private View layout_roll_view;
    /*
     * 这里的 title 是轮播图中图片的介绍说明而不是当前页面头部的标题
     */
    private List<String> titleList = new ArrayList<String>();
    /*
     * 轮播图中的每个图是从网上动态加载的，因此我们把这些图片地址也放到集合中
     */
    private List<String> urlImgList = new ArrayList<String>();

    /*
     * 轮播图下面的点点也放到集合中
     */
    private List<View> viewList = new ArrayList<View>();
    /*
     * 用于放轮播图，对的，轮播图是放在线性布局中的
     */
    @ViewInject(R.id.top_news_viewpager)
    private LinearLayout top_news_viewpager;
    /*
     * 轮播图图片标题
     */
    @ViewInject(R.id.top_news_title)
    private TextView top_news_title;

    /*
     * 轮播图的点点
     */
    @ViewInject(R.id.dots_ll)
    private LinearLayout dots_ll;
    /*
     * 自定义（引用第三方）的 ListView 对象
     */
    @ViewInject(R.id.lv_item_news)
    private PullToRefreshListView ptrlv;
    //ListView 的适配器
    private MyBaseAdapter myBaseAdapter;
    private String moreUrl;
    // 放置底部 item 条目数据的集合的
    private List<News> newList = new ArrayList<News>();
    private List<String> idList = new ArrayList<String>();
    protected static final String IDS = "ids";
    //构造函数中的 url 为当前页面数据地址
```

```
    public NewItemPager(Context context,String url) {
        super(context);
        this.url = url;
    }
    /*
     * 在 initView 方法中初始化了两个布局文件
     * 一个是轮播图
     * 一个是 ListView
     */
    @Override
    public View initView() {
        layout_roll_view = View.inflate(context, R.layout.layout_roll_view, null);
        ViewUtils.inject(this, layout_roll_view);
        top_news_viewpager = (LinearLayout)
layout_roll_view.findViewById(R.id.top_news_viewpager);

        view = View.inflate(context, R.layout.frag_item_news, null);
        ViewUtils.inject(this, view);
        ptrlv = (PullToRefreshListView) view.findViewById(R.id.lv_item_news);

        // 下拉加载的事件屏蔽
        ptrlv.setPullLoadEnabled(false);
        // 包含下拉刷新，上拉加载操作
        ptrlv.setScrollLoadEnabled(true);
        /*
         * 这个刷新监听是自定义控件类 PullToRefreshListView 自带的
         */
        ptrlv.setOnRefreshListener(new OnRefreshListener<ListView>() {

            @Override
            public void onPullDownToRefresh(PullToRefreshBase<ListView> refreshView) {
                /*
                 * 把 ListView 往下拉时刷新数据
                 * 获取新数据
                 */
                getNewItemPager(url, true);
            }

            @Override
            public void onPullUpToRefresh(PullToRefreshBase<ListView> refreshView) {
                /*
                 *  把 ListView 往上拉时加载更多
                 */
```

```
                getNewItemPager(moreUrl, false);
            }
        });
        /*
         * 新闻条目被点击时的处理事件
         * 处理了 2 中业务
         * 1、把第一次读到的新闻背景变色
         * 2、通过显示意图打开一个 Activity，显示新闻内容
         */
        ptrlv.getRefreshableView().setOnItemClickListener(new OnItemClickListener()
{
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3)
{
                // 如果已经是度过的新闻，则不需要再进行存储操作
                if (!newList.get(arg2 - 1).isRead) {
                    /*
                     * 设置为已读
                     */
                    newList.get(arg2 - 1).isRead = true;
                    //将新闻的 id 存储在 xml 中
                    String ids = SharedPreferencesUtil.getStringData(context, IDS, "");
                    SharedPreferencesUtil.saveStringData(context, IDS, ids + "#" +
newList.get(arg2 - 1).id);
                }
                //通知 ListView 的 Adapter 更新 ListView
                myBaseAdapter.notifyDataSetChanged();
                //打开一个 Activity，显示新闻内容
                Intent intent = new Intent(context, NewsDetailActivity.class);
                intent.putExtra("url", newList.get(arg2 - 1).url);
                context.startActivity(intent);
            }
        });
        return view;
    }

    @Override
    public void initData() {
        // 现将存储在本地的 id 分割成一个数组，
        // 并且放置到结合中，然后跟请求服务端的数据进行匹配(id)，如果 id 匹配上了，则说明已
读，将服务端返回数据 News 对象中 isRead 字段设置成 true,否则 false
        idList.clear();
        String ids = SharedPreferencesUtil.getStringData(context, IDS, "");
```

```
        String[] strings = ids.split("#");

        for (int i = 0; i < strings.length; i++) {
            idList.add(strings[i]);
        }
        String result = SharedPreferencesUtil.getStringData(context, HMApi.BASE_URL +
url, "");
        if (!TextUtils.isEmpty(result)) {
            processData(result, true);
        }else{
            getNewItemPager(url, true);
        }
    }

    private void getNewItemPager(final String url, final boolean isRefresh) {
        if (!TextUtils.isEmpty(url)) {
            // 获取当前页数据操作
            getdata(HttpMethod.GET, HMApi.BASE_URL + url, null, new
RequestCallBack<String>() {
                @Override
                public void onSuccess(ResponseInfo<String> responseInfo) {
                    SharedPreferencesUtil.saveStringData(context, HMApi.BASE_URL + url,
responseInfo.result);
                    processData(responseInfo.result, isRefresh);
                }
                @Override
                public void onFailure(HttpException error, String msg) {
                }
            });
        } else {
            Toast.makeText(context, "没有更多数据", 0).show();

            // 隐藏顶部底部的刷新加载条目
            ptrlv.onPullDownRefreshComplete();
            ptrlv.onPullUpRefreshComplete();
        }
    }

    // 解析 json 操作
        private void processData(String result, boolean isRefresh) {
            NewBean bean = GsonUtil.jsonToBean(result, NewBean.class);
            // 将当前对应 bean 中的轮播图的 title，url 单独获取出来，(选中，未选中状态)点
            // 如果返回状态码为 200
```

```
        if (bean.retcode.equals("200")) {
            //加载更多链接
            moreUrl = bean.data.more;
            //如果轮播图中的新闻数据大于 0
            if (bean.data.topnews.size() > 0) {
                if (isRefresh) {
                    titleList.clear();
                    urlImgList.clear();
                    for (int i = 0; i < bean.data.topnews.size(); i++) {
                        titleList.add(bean.data.topnews.get(i).title);
                        urlImgList.add(bean.data.topnews.get(i).topimage);
                    }

                    initDot();

                    // 组装(将对应的 view 剖析出来，以后按照传递参数的方式直接去使用)
                    RollViewPager rollViewPager = new RollViewPager(context,
viewList, new RollViewPager.onPageClick() {
                        @Override
                        public void onclick(int i) {
                            Toast.makeText(context, "position = " + i, 0).show();
                        }
                    });
                    rollViewPager.initTitleList(top_news_title, titleList);
                    rollViewPager.initImgUrlList(urlImgList);
                    rollViewPager.startRoll();

                    top_news_viewpager.removeAllViews();
                    top_news_viewpager.addView(rollViewPager);

                    // 需要添加到 listView 上面去
                    // layout_roll_view

                    // 就是个 listView
                    if (ptrlv.getRefreshableView().getHeaderViewsCount() < 1) {
                        ptrlv.getRefreshableView().addHeaderView(layout_roll_view);
                    }
                }
            }
            // 填充 listView 后头部轮播图才会显示
            if (bean.data.news.size() > 0) {
                //如果只是刷新则把旧数据清空
                if (isRefresh) {
```

```
                // 如果刷新则将原有数据全部清空，然后进行添加操作
                newList.clear();
            }
            // 在原有基础上做添加操作
            newList.addAll(bean.data.news);
            /*
             * 遍历新闻，设置是否已读标识
             */
            for (int i = 0; i < newList.size(); i++) {
                if (idList.contains(newList.get(i).id)) {
                    newList.get(i).isRead = true;
                } else {
                    newList.get(i).isRead = false;
                }
            }

            // 填充 listView
            if (myBaseAdapter == null) {
                myBaseAdapter = new MyBaseAdapter(context, newList);
                ptrlv.getRefreshableView().setAdapter(myBaseAdapter);
            } else {
                myBaseAdapter.notifyDataSetChanged();
            }
        }

        // 隐藏顶部底部的刷新加载条目
        ptrlv.onPullDownRefreshComplete();
        ptrlv.onPullUpRefreshComplete();
    }
}

class MyBaseAdapter extends HMAdapter<News> {

    public MyBaseAdapter(Context context, List<News> list) {
        super(context, list);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            convertView = View.inflate(context, R.layout.layout_news_item, null);
        }
        /*
```

```
            * 这里调用 xUtils 框架中的 BitMapUtils，来从网络上加载图片
            * 该框架可以解决大图片加载，图片缩放等问题
            */
           BitmapUtils bitmapUtils = new BitmapUtils(context);
           //给新闻设置属性
           ImageView iv_img = (ImageView) convertView.findViewById(R.id.iv_img);
           TextView tv_title = (TextView) convertView.findViewById(R.id.tv_title);
           TextView tv_pub_date = (TextView)
convertView.findViewById(R.id.tv_pub_date);
           //给 iv_img 控件加载图片
           bitmapUtils.display(iv_img, list.get(position).listimage);
           tv_title.setText(list.get(position).title);
           tv_pub_date.setText(list.get(position).pubdate);
           //如果已读则给新闻背景变色
           if (list.get(position).isRead) {

   tv_title.setTextColor(context.getResources().getColor(R.color.news_item_has_re
ad_textcolor));
           } else {

   tv_title.setTextColor(context.getResources().getColor(R.color.news_item_no_rea
d_textcolor));
           }
           return convertView;
       }
   }
   /*
    * 初始化轮播图的点点们
    */
   private void initDot() {
       dots_ll.removeAllViews();
       viewList.clear();

       for (int i = 0; i < urlImgList.size(); i++) {
           View view = new View(context);
           if (i == 0) {
               view.setBackgroundResource(R.drawable.dot_focus);
           } else {
               view.setBackgroundResource(R.drawable.dot_normal);
           }

           LinearLayout.LayoutParams          layoutParams          =          new
LinearLayout.LayoutParams(CommonUtil.dip2px(context, 6),
```

```
CommonUtil.dip2px(context, 6));
            view.setLayoutParams(layoutParams);
            layoutParams.setMargins(5, 0, 5, 0);
            dots_ll.addView(view);
            viewList.add(view);
        }
    }
}
```

在上面的代码中我们将新闻具体对象封装在 NewBean 中，其代码如下所示：

3.3.5.4 NewBean

```
package com.itheima.zhbj_heima.bean;

import java.util.List;

public class NewBean {
    public NewBeanItem data;
    public String retcode;

    public class NewBeanItem{
        public String countcommenturl;
        //上拉加载
        public String more;
        public String title;

        public List<News> news;
        public List<Topic> topic;
        public List<Topnews> topnews;
    }

    public class News{
        public String comment;
        public String commentlist;
        public String commenturl;
        //唯一性标志当前新闻条目
        public String id;
        //新闻列表的图片
        public String listimage;
        //时间
```

3.3.5.5 RollViewPager

在 NewItemPager 类中我们用到了自定义额的 ViewPager ，RollViewPager，其代码清单如下：

```
public class RollViewPager extends ViewPager {
    private Context context;
    /*
     * ViewPager 中的每一页
     */
    private List<View> viewList;
    /*
     * 顶部标题（中国 背景 社会等等）
     */
    private TextView top_news_title;
    /*
     * 轮播图标题
     */
    private List<String> titleList;
    /*
     * 轮播图链接地址
     */
    private List<String> urlImgList;
    private BitmapUtils bitmapUtils;
    /*
     * ViewPager 适配器
     */
    private MyAdapter myAdapter;
    //定时任务，任务内容就是轮播图片
    private RunnableTask runnableTask;
    //当前轮播指针（脚标）
    private int currentPosition = 0;

    private Handler handler = new Handler(){
        public void handleMessage(android.os.Message msg) {
            //处理了滑动
            RollViewPager.this.setCurrentItem(currentPosition);
            //开始下次滚动
            startRoll();
        };
    };
    class RunnableTask implements Runnable{
        @Override
        public void run() {
```

```
            //滚动 viewpager
            currentPosition = (currentPosition+1)%viewList.size();
            handler.obtainMessage().sendToTarget();
        }
    }
    private int downX;
    private int downY;
    /*
     * 当前源文件中定义的 OnPageClick 接口
     * 接口中定义一个 onPageClick 方法
     * 当用户点击了 ViewPager 的一页的处理逻辑由调用该类的
     * 调用者处理
     */
    private OnPageClick pageClick;
    /*
     * 覆写父类的事件分发函数
     * 当手指按下 ViewPager 中的某个 Pager 的时候
     */
    @Override
    public boolean dispatchTouchEvent(MotionEvent ev) {
        switch (ev.getAction()) {
        case MotionEvent.ACTION_DOWN:
            /*
             * 让当前 ViewPager 对应的控件不要去拦截事件
             * 也就是 ACTION_DOWN 事件传递给子控件（就是 ViewPager 中的某一个 Pager）
             */
            getParent().requestDisallowInterceptTouchEvent(true);
            downX = (int) ev.getX();
            downY = (int) ev.getY();
            break;
        case MotionEvent.ACTION_MOVE:
            int moveX = (int)ev.getX();
            int moveY = (int)ev.getY();

            /*
             * 如果是上下滑动则拦截该事件，让该事件不在 ViewPager 的某一个 Pager 上发生
             */
            if(Math.abs(moveY-downY)>Math.abs(moveX-downX)){
                getParent().requestDisallowInterceptTouchEvent(false);
            }else{
                /*
                 * 如果是水平方向滑动，则让该事件发生在 ViewPager 控件上，ViewPager 接收到了
就实现切换页面
```

```
                */
                getParent().requestDisallowInterceptTouchEvent(true);
            }
            break;
        }
        return super.dispatchTouchEvent(ev);
    };

    //从界面移出的时候会调用方法
    @Override
    protected void onDetachedFromWindow() {
        super.onDetachedFromWindow();
        //移除所有的任务
        handler.removeCallbacksAndMessages(null);
    }

    public RollViewPager(Context context, final List<View> viewList,OnPageClick
pageClick) {//new RollViewPager.onPageClick()
        super(context);
        this.context = context;
        this.viewList = viewList;
        this.pageClick = pageClick;
        bitmapUtils = new BitmapUtils(context);
        runnableTask = new RunnableTask();

        this.setOnPageChangeListener(new OnPageChangeListener() {

            @Override
            public void onPageSelected(int arg0) {
                top_news_title.setText(titleList.get(arg0));
                for(int i=0;i<urlImgList.size();i++){
                    if(i==arg0){

    viewList.get(arg0).setBackgroundResource(R.drawable.dot_focus);
                    }else{
                        viewList.get(i).setBackgroundResource(R.drawable.dot_normal);
                    }
                }
            }

            @Override
            public void onPageScrolled(int arg0, float arg1, int arg2) {
                // TODO Auto-generated method stub
```

```
        }

        @Override
        public void onPageScrollStateChanged(int arg0) {
            // TODO Auto-generated method stub

        }
    });
}
//将图片关联说明的文字集合,需要显示的控件传递进来
public void initTitleList(TextView top_news_title, List<String> titleList) {
    if(null != top_news_title && null != titleList && titleList.size()>0){
        top_news_title.setText(titleList.get(0));
    }
    this.top_news_title = top_news_title;
    this.titleList = titleList;
}
//显示图片的 url 地址的集合
public void initImgUrlList(List<String> urlImgList) {
    this.urlImgList = urlImgList;
}

public interface OnPageClick{
    public abstract void onclick(int i);
}

public void startRoll() {
    //滚动 viewpager
    if(myAdapter == null){
        myAdapter = new MyAdapter();
        this.setAdapter(myAdapter) ;
    }else{
        myAdapter.notifyDataSetChanged();
    }
    //延时 3 秒执行任务
    handler.postDelayed(runnableTask, 3000);
}

class MyAdapter extends PagerAdapter{
    @Override
    public int getCount() {
        return urlImgList.size();
    }
```

```
        @Override
        public boolean isViewFromObject(View arg0, Object arg1) {
            return arg0 == arg1;
        }

        @Override
        public Object instantiateItem(ViewGroup container, final int position) {
            View view = View.inflate(context,R.layout.viewpager_item, null);
            ImageView imageView = (ImageView) view.findViewById(R.id.image);
            bitmapUtils.display(imageView,urlImgList.get(position));

            view.setOnTouchListener(new OnTouchListener() {
                private int downX;
                private long downTime;

                @Override
                public boolean onTouch(View v, MotionEvent event) {
                    switch (event.getAction()) {
                    case MotionEvent.ACTION_DOWN:
                        /*
                         * 取消 handler 队列中的待执行任务
                         * 实现的效果就是轮播图片暂停轮播
                         */
                        handler.removeCallbacksAndMessages(null);//按住图片的时候移出图
片的 轮播方法

                        downX = (int) event.getX();
                        downTime = System.currentTimeMillis();
                        break;
                    case MotionEvent.ACTION_UP:
                        /*
                         * 如果 keyDown 和 KeyUp 之间的时间小于 0.5 秒，并且 x 坐标也一直，则认
为是单击事件
                         */
                        if(System.currentTimeMillis()-downTime<500 && downX ==
event.getX()){
                            //点击事件被触发
                            if(pageClick!=null){
                                pageClick.onclick(position);
                            }
                        }
                        startRoll();
                        break;
```

```
                case MotionEvent.ACTION_CANCEL:
                    startRoll();
                    break;
                }
                return true;
            }
        });

        container.addView(view);
        return view;
    }

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

3.4 MyViewPager

3.4.1 MyViewPager

MyViewPager 类很简单只是单纯的继承了父类 LazyViewPager，同时覆写了父类的事件分发拦截器方法，在该方法中我们返回 false 这样保证了我们 ViewPager 不拦截子控件的触摸事件。如果 onInterceptTouchEvent 返回 true 会是什么效果呢？如果返回 true 那么我们在子控件上的滑动事件被父控件也就是我们的 ViewPager 给拦截了，并且该事件发生在了自己身上，那么我们的 ViewPager 就切换页面了，而根据我们的逻辑我们这里的 ViewPager 中的每一个页面代表一个一级菜单底部的 tab 内容，不应该换页。

```
/**
 * 这里之所以继承 LazyViewPager
 * 是因为我们需要覆写父类的 onInterceptTouchEvent 方法
 * 该方法返回 false 就是让子控件的触摸事件分发出去
 * 返回 true 则对子控件的触摸事件进行拦截
 */
public class MyViewPager extends LazyViewPager {
```

```
    public MyViewPager(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    public MyViewPager(Context context) {
        super(context);
    }
    //返回 false 则
    @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {
        return false;
    }

    @Override
    public boolean onTouchEvent(MotionEvent ev) {
            return false;
    }
}
```

3.4.2 LazyViewPager

这里用到的 LazyViewPager 是 Github 上的一个开源框架，我们直接把源码放到我们的代码中进行使用，这样我们还能对其进行修改。我们知道我们这里的 MyViewPager 中的每一页是一个一级菜单项,Android 原生的 ViewPager 是默认初始化我们当前页的前后各一页（如果有的话）的，而这不是我们想要的，根据我们的业务逻辑，当用户选择哪个一级菜单额时候我们在加载哪个 Pager，同时初始化该 Pager 对应的左侧 SlidingMenu 内容，如果用原生的 ViewPager 则把我们 SlidingMenu 给搞晕了，因为 SlidingMenu 只有一个，每初始化一个 Pager 就会去修改 SlidingMenu，那么当我们同时初始化 3 个 Pager 的时候，SlidingMenu 怎么处理呢，显然复杂化了，因此我们使用 LazyViewPager，一次只初始化一个 Pager。

LazyViewPager 是开源框架 ,因此这里没有必要贴出其源码 ,只有一个地方需要说明 ,就是我们的 LazyViewPager 有个参数是指定初始化几个 Pager 的，我们需要改为 0，也就是只初始化我们当前使用到的 Pager。

我们需要修改的内容截图如下所示：

```
public class LazyViewPager extends ViewGroup {
    private static final String TAG = "LazyViewPager";
    private static final boolean DEBUG = false;

    private static final boolean USE_CACHE = false;
    //默认多初始化几个pager，我们设置为0，也就是只加载当前页
    private static final int DEFAULT_OFFSCREEN_PAGES = 0;
```

4、菜单 MenuFragment（★★★）

我们项目的另外一条主线就是 MenuFragment，不过幸好比较简单。

4.1 MenuFragment 布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#262626"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/tv_menu_classify"
        android:layout_width="match_parent"
        android:layout_height="55dp"
        android:padding="3dp"
        android:text="分类"
        android:textColor="@color/white"
        android:textSize="18sp"
        android:visibility="invisible" />

    <!-- 三个 listView 作用， -->
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <ListView
            android:id="@+id/lv_menu_news_center"
            android:layout_width="match_parent"
```

```
            android:layout_height="match_parent"
            android:cacheColorHint="@android:color/transparent"
            android:divider="@null"
            android:fadingEdge="none"
            android:focusable="true"
            android:listSelector="@drawable/transparent"
            android:scrollbars="none"
            android:visibility="visible" >
        </ListView>

        <ListView
            android:id="@+id/lv_menu_smart_service"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:cacheColorHint="@android:color/transparent"
            android:divider="@null"
            android:fadingEdge="none"
            android:focusable="true"
            android:listSelector="@drawable/transparent"
            android:scrollbars="none"
            android:visibility="gone" >
        </ListView>

        <ListView
            android:id="@+id/lv_menu_govaffairs"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:cacheColorHint="@android:color/transparent"
            android:divider="@null"
            android:fadingEdge="none"
            android:focusable="true"
            android:listSelector="@drawable/transparent"
            android:scrollbars="none"
            android:visibility="gone" >
        </ListView>
    </FrameLayout>

</LinearLayout>
```

Tips：上面的布局整体是 LinearLayout，里面的 FrameLayout 包含了 3 个 LinearLayout，分别是新闻中心，智慧服务，政务指南，其他 2 个 tab 之所有没有是因为他们不需要。

4.2 MenuFragment 代码

```
public class MenuFragment extends BaseFragment {
    private static final String tag = "MenuFragment";
    /*
     * 左侧菜单的条目名称
     */
    private List<String> titleList;

    @ViewInject(R.id.lv_menu_news_center)
    public ListView lv_menu_news_center;
    private MyAdapter adapter;
    private int currentPosition = 0;

    @Override
    public View initView(LayoutInflater inflater) {
        view = inflater.inflate(R.layout.layout_left_menu, null);
        ViewUtils.inject(this, view);
        return view;
    }
    @Override
    public void initData(Bundle savedInstanceState) {

    }
    //将底部条目关联过来的数据去填充左边条目
    public void initMenu(List<String> titleList) {
        this.titleList = titleList;
        Log.i(tag, "titleList.size()="+titleList.size());
        adapter = new MyAdapter(context,titleList);
        lv_menu_news_center.setAdapter(adapter);

        lv_menu_news_center.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,long arg3)
{
                currentPosition = arg2;
                adapter.notifyDataSetChanged();

                //在此处要去调用，NewCenterPager 对象中的 switchPager（i）方法
                //1，获取 NewCenterPager 对象
                //2，NewCenterPager 对象来至于 HomeFragment 在生成 5 个页面(向 list 结合中添
加了 5 个对象)过程中
```

```
            //3，HomeFragment 对应对象获取出来？
            //4，HomeFragment 在 MainActivity

   ((MainActivity)context).switchHomeFragment().switchNewCenterPager().switchPage
r(arg2);
            slidingMenu.toggle();
         }
      });
   }

   class MyAdapter extends HMAdapter<String>{
      public MyAdapter(Context context, List<String> list) {
         super(context, list);
      }
      @Override
      public View getView(int position, View convertView, ViewGroup parent) {
         ViewHolder holder = null;
         if(convertView == null){
            holder = new ViewHolder();
            convertView = View.inflate(context, R.layout.layout_item_menu, null);

            holder.iv_menu_item = (ImageView)
convertView.findViewById(R.id.iv_menu_item);
            holder.tv_menu_item = (TextView)
convertView.findViewById(R.id.tv_menu_item);

            convertView.setTag(holder);
         }else{
            holder = (ViewHolder) convertView.getTag();
         }
         holder.tv_menu_item.setText(titleList.get(position));

         //如果不添加 else 则选中不可返回原有样式
         if(position == currentPosition){
            holder.iv_menu_item.setImageResource(R.drawable.menu_arr_select);

   holder.tv_menu_item.setTextColor(context.getResources().getColor(R.color.red))
;
            convertView.setBackgroundResource(R.drawable.menu_item_bg_select);
         }else{
            holder.iv_menu_item.setImageResource(R.drawable.menu_arr_normal);
   holder.tv_menu_item.setTextColor(context.getResources().getColor(R.color.white
));
```

```
            //透明
            convertView.setBackgroundResource(R.drawable.transparent);
        }
        return convertView;
    }
}

class ViewHolder{
    ImageView iv_menu_item;
    TextView tv_menu_item;
}
}
```

Tips：MenuFragment 的逻辑比较简单，只需要拿到 initMenu 中的 titleList 把这些数据填充到 ListView 中，然后告知 NewsCenterPager 切换对应页，然后隐藏就可以了。

5、项目下载（0）

我将自己的项目打包放在百度云盘上供大家下载：

http://pan.baidu.com/s/1pJl0auB

至此，本文档完！

2015 年 3 月 13 日 星期五 16:50:15

北京市海淀区中关村软件园国际软件大厦