# 导游宝界面设计详细文档

## 文档主旨

该文档是导游宝界面设计的详细说明文档，主要讲述了导游宝中各个界面的详细设计、实现细节。

## 导游宝界面一览

1. 跳转界面（欢迎/导航界面）
2. 城市选择界面
3. 博物馆选择界面
4. 主界面

主界面包括4个子界面

1. 博物馆简介界面
2. 随行导游界面
3. 专题导游界面
4. 地图导游界面
5. 侧边栏界面
6. 下载中心界面
7. 搜索界面
8. 设置界面
9. 对话框

## 各个界面详细设计

该部分采用**由总到分**的方式进行讲解，即先描述界面的总体设计（由哪些控件组成），再详细讲解组成该界面的各个控件的具体实现细节。

### 跳转界面（欢迎/向导界面）

该界面目前只实现了跳转功能，未实现具体界面显示。实现具体界面时，可以使用一个ImageView展示一张导游宝主题图片，作为欢迎页；或者是使用一个ViewPager实现APP向导界面，具体实现可参考本文档 “博物馆简介界面”中的“博物馆图片展示控件的实现”。

跳转功能：在跳转界面创建完成后，即onCreate()回调方法被调用时，开启一个新线程，在线程休眠一段时间后(程序中是1000ms)，使用android消息处理机制，向主线程发送一条页面跳转的message。主线程的handler获得该message后完成页面跳转。

### 城市选择界面

城市选择界面主要包括三个部分——百度地图定位模块+快速定位侧边栏+城市列表（ListView）。

主要难点在于快速定位侧边栏的实现，及设置其与城市列表的联动。

来看各部分具体实现：

#### 百度地图API定位使用

1. 设置自己申请的key，即ak值，在application标签下进行添加
2. 调用百度lbs的service
3. 在Application下进行初始化SDKInitializer.initialize(this);
4. 首先创建自己的LocationClient进行定位，并设置定位监听的回调接口，在回调接口中获得位置信息
5. 调用LocationClient的start方法开始定位，在onDestory方法中调用stop方法停止定位

以上部分可以参考博客：

http://blog.csdn.net/smbroe/article/details/41441495

#### 快速定位侧边栏实现：QuickLocationBar类

1. 字母绘制，根据总的高度除以字符串数组的长度来得到每一个字符的高度，然后循环遍历整个数组在onDraw方法中绘制字符；
2. 处理触摸事件：通过复写父类的dispatchTouchEvent方法来实现，首先为写一个回调接口负责在触摸位置的字符改变时进行调用，当发生按下事件或移动事件时，我们根据触摸点的位置计算出当前触摸的字符，如果和我们显示的字符不相同则执行回调接口的方法，并进行View的重绘。在这里

int c = (int) (y / getHeight() \* characters.length)即表示触摸的字母的下标。

#### 城市列表的实现

//TODO 修改

首先从网络上搜到了一个数据库.db的文件，该数据库包含了中国各城市的中文名称以及首字母。将该数据库文件拷贝到我们的工程目录的res/raw文件 夹之下（该文件夹不会被Android编译），当我们访问数据库时首先检查该数据库文件是否存在（一般数据库文件会被Android系统放在/data /data/packagename/databases文件夹之下），如果不存在则将数据库拷贝过来。

以上部分可以参考博客：

http://blog.csdn.net/smbroe/article/details/43058455

### 博物馆选择界面

该界面由两部分组成：界面导航栏TopBar+博物馆列表（ListView）

在开启该界面的Intent对象中传入了一个city值，记录了当前所在城市。该界面通过传入的city值获取相应的博物馆列表。

在线模式下，博物馆列表通过实时API ：城市博物馆列表API获得数据解析得来。解析出来的列表，与本地已下载的离线包做匹配，如果本地存在离线包，则标记为已有数据，否则标记为无；

离线模式下，博物馆列表只显示该城市中已下载的博物馆列表，由本地数据库中获取。

#### 界面头部(TopBar)的实现

**自定义View——TopBar**

几乎每一款APP都会自定义自己的TopBar用以作为APP的导航栏，而不使用Android提供的ActionBar/Toolbar（满足Matrix Design规范的APP除外），虽然Google日渐提高ToolBar使用的灵活性，但还是不如自定义的灵活。自定义TopBar的好处就是定制性强，后期可根据需求做相应的更改。下面我们来看如何实现：

View组成——由3部分组成，左右两个ImageView+中间一个TextView。如图：



左边ImageView：ivMenu 菜单按钮，用于调出侧边栏

右边ImageView：ivSearch 搜索按钮，用于跳转到搜索界面

TextView：标题栏，用于显示标题

主要是需要给两个ImageView设置监听事件。比较简单不再说明

此外，由于有的界面，如随行导游是不需要搜索按钮的，所以TopBar需要向外暴露一个接口，用以设置搜索按钮的可见性。

public void setSearchingVisible**(**boolean visible**){**

**if(**visible**)**

ivSearch**.**setVisibility**(**View**.**VISIBLE**);**

**else**

ivSearch**.**setVisibility**(**View**.**GONE**);**

**}**

同时，考虑到有的界面的标题栏是显示在左边的，所以需要再暴露一个接口可以设置标题栏的对齐属性

//设置左对齐

public void setTitleLeftGravity(int gravity){

tvTitle.setGravity(Gravity.*LEFT*);

tvTitle.setGravity(Gravity.*CENTER\_VERTICAL*);

}

然后就是其他的一些常用设置属性的方法，比如设置TopBar背景，设置标题文字等。

#### 博物馆列表

就是给ListView设置Adapter，Adapter中根据item布局文件（.xml）和List<MuseumBean>的数据设置各个view的数据并显示。

值得一提的是，项目中为了简化代码，提高复用性，实现了**通用适配器CommonAdapter**，我们来看看CommonAdapter的实现：

**通用适配器CommonAdapter与万能ViewHolder**

主要工作就是归纳总结一般情况下为ListView/GridView配置适配器时Adapter需要编写的代码，将其高度抽象，并尽可能地提炼出其中可复用的成分，使得之后书写相同代码时（配置Adapter时）只需书写最少量的代码。

具体实现见代码.adapter.CommonAdapter类和.adapter.ViewHolder类。

可参考博客：

<http://blog.csdn.net/lmj623565791/article/details/38902805>

可扩展ListViewAdapter类: DownloadListFragment.java

### 主界面

该界面为导游宝主要功能界面，也是与用户交互最多的“一个”界面。该界面根据系统功能共分为4个子界面（4个Fragment），分别是博物馆简介界面、随行导游界面、专题导游界面和地图导游界面。

使用了RadioGroup（底部导航栏）+FragmentManager的方式去管理4个子界面的显示和切换。通过RadioGroup#setOnCheckedChangeListener(listener)实现RadioButton点击与页面切换的联动以及控制页面切换逻辑。

具体实现见.adapter.FragmentTabAdapter类

下面我们来看看各个子界面的页面构成和实现细节

#### 博物馆简介界面

该界面主要由3部分构成——博物馆图片展示部分+博物馆文字简介+精品列表（上拉加载更多ListView）

##### 博物馆图片展示控件的实现——ViewPager+LinearLayout(导航小圆点)

这个实现比较简单，只是ViewPager的一般用法的基础上加上导航小圆点而已。

导航小圆点就是一个个ImageView，用两种图标分别表示选中和未选中（实心和空心），放在属性android:orientation=”horizontal”的一个LinearLayout中。并在程序中根据ViewPager中图片的个数，动态添加相同个数的圆点，设置与ViewPager的联动即可——如ViewPager选中第一张图片时，第一个圆点为实心，其他未选中为空心等。

**需要注意的是**，由于ViewPage会自动占满整个屏幕，所以需要我们在代码中动态设置ViewPager的高度。如下：

FrameLayout**.**LayoutParams params **=** **new** FrameLayout**.**LayoutParams**(**

FrameLayout**.**LayoutParams**.**MATCH\_PARENT**,** dm**.**heightPixels **\*** 2 **/** 5**);**

viewPager**.**setLayoutParams**(**params**);**

其中dm是屏幕像素矩阵对象（DisplayMetrics），通过dm可以获取屏幕的宽、高等信息。dm设置过程如下：

DisplayMetrics dm = **new** DisplayMetrics();

**this**.getActivity().getWindowManager().getDefaultDisplay().getMetrics(dm);

##### 博物馆文字简介

这部分也比较简单，就是一个TextView(文字展示)+一个ImageView实现点击语音播放（MediaPlayer）

##### 上拉加载更多ListView实现——AutoLoadListView

1. 首先，为ListView添加footer布局，用于显示正在加载、加载完成、加载失败的信息
2. 设置回调接口OnLoadListener，该回调接口中共有两个方法，onLoad方法用于执行加载更多的方法，onRetry方法用于执行加载失败时的重新加载。
3. 设置标志isLoading表示是否正在加载，在滑动事件中监听是否滑动到底部，若滑动到底部（view.getLastVisiblePosition() == view.getCount() - 1）且isLoading为false，设置footer显示正在加载界面，设置isLoading为true，同时执行回调接口中的onLoad方法。加载完成后需要调用onLoadcomplete方法通知AutoLoadListView加载完成。
4. 若加载失败，则调用setLoadFailed方法显示加载失败的界面。点击加载失败的界面会调用回调接口的onRetry方法重新加载数据。

方法解释

1.setOnLoadListener(OnLoadListener onLoadListener)--设置加载更多监听事件

2.setOnMyScrollListener(OnScrollListener onScrollListener) --设置滑动监听事件

3.public interface OnLoadListener {

public void onLoad();

public void onRetry();

}

加载更多回调接口，onLoad方法执行加载更多方法，onRetry执行加载失败后点击重新进行加载事件。

#### 随行导游界面

该界面是整个应用中最为复杂的一个界面，因为这个界面需要考虑许多业务逻辑，所以实现起来比较困难。

该界面主要由3个部分构成——歌词显示部分（LyricView）+播放栏+图片展示栏，但其实3部分各自的实现都不难，最难的是为这三部分设置联动。

先看看各个部分的实现：

##### 歌词联动View的实现--LyricView

1. 读取歌词文件：LyricObject记录每一句歌词的信息，lrc\_map用来存储整篇歌词信息，其key表示该句歌词开始时间（毫秒为单位），value是LyricObject

以该句歌词为例：

1）首先取出中括号（[]）中的内容00位置表示分，28表示秒，91表示毫秒位，我们将其转化为一个int类型值以毫秒为计数单位int currTime = (m \* 60 + s) \* 1000 + ms \* 10;

2）接下来读取该句歌词并创建一个新的LyricObject对象存储该句歌词信息，

3）将信息添加到map中：lrc\_read.put(currTime, item1);

4）遍历lrc\_read，记录每句歌词的结束时间，并计算出整句歌词的用时，最后将信息完善的LyricObject对象放置在lrc\_map中

1. 准备工作--prepare方法
2. 创建MediaPlayer，用于播放音频文件
3. 读取歌词文件
4. 设置显示字体大小，以篇歌词中的最长句确定歌词字体的大小wordSize = (int) (width / max /1.5);
5. 开始播放
6. 创建异步线程，每隔100ms更新一次，用于计算当前歌曲中正在播放的那一句：调用mediaPlayer.getCurrentPosition()获得当前播放时间，遍历lrc\_map，获取每句歌词的开始时间，若小于当前时间则index加一，否则跳出循环，那么当前播放的句子为index-1。

计算歌词滚动的速度，即歌词向下平移的距离：offsetY表示歌词在Y轴上的偏移量，随着时间增加，该值会减小，SIZE\_TEXT\_DISPLAY表示当前播放句应当距离顶部的长度，offsetY + (wordSize + INTERVAL) \* lrcIndex表示当前播放句实际距离顶部的距离，两者只差就表示需要移动的距离。

1. 绘制歌词，首先绘制当前播放的句的歌词，接着绘制当前之前的歌词以及当前之后的歌词
2. 触摸事件：在onTouch方法中对offsetY进行改变

方法解释：

1. setCurrentPosition(int position)--设置当前播放时间
2. prepare(String mp3Path, String lyricPath)--设置音频文件地址以及歌词文件地址
3. pause()--暂停播放
4. stop()--停止播放
5. isPlaying()--查询是否正在播放

##### 播放栏的实现

播放栏的实现比较简单，就是一个原生的水平风格progressBar+3个ImageView,分别为3个ImageView设置点击监听即可。不再详述

##### 图片展示栏的实现

图片展示栏包含了两个部分——标题栏+图片展示部分。

这里实现了抽屉风格的展示效果，即可以通过点击设置内容可见或隐藏。如图：





抽屉效果的实现很简单，就是监听图片展示部分的可见性，然后给TextView（标题栏）设置点击监听，当图片展示部分不可见时，点击textView，设置图片部分可见，且让右边的ImageView(箭头)变换背景图片。同理当图片不可见时，点击时设置图片可见。不再详述。

主要来看图片展示部分的实现。这里采用自定义HorizontalScrollView 实现图片展示View。并且使用动态加载的方式,保证同一时刻只加载<=X张图片（X由手机的屏幕宽度决定），有效避免程序出现OOM（图片绘制会消耗大量的内存，当绘制的图片过多时，会出现内存不足Out of Memory的系统错误）。

**图片展示View——GalleryView（继承自HorizontalScrollView）**

1. View构成：包含唯一一个子View——LinearLayout.在LinearLayout下是一个个ImageView（用于显示图片），我们实现对LinearLayout中每个ImageView的加载与显示，就可以实现图片的动态加载与展示了。
2. 需要实现功能：
3. 使用Adapter实现图片与ImageView的适配。
4. 监听滚动事件，动态创建/销毁图片(ImageView)，实现每次只加载X张图片
5. 设置图片(Item，每一张图片就相当于GalleryView中的一个Item)的点击事件，实现点击接口回调，以及点击时GalleryView滚动到相应位置
6. 定义加载更多接口，监听滚动事件，当滚动到最后一张图片/第一张图片时触发接口回调，实现加载更多。如果没有设置加载更多接口，则循环加载图片。
7. 实现

根据需要实现功能列表一步步实现即可。

1. 实现一个Adapter,用以实现image(图片)到imageView间的转换。这里使用了NetworkImageView——Volley封装的一种ImageView。可以使用Volley封装的带有缓存机制的ImageLoader去加载网络图片。还可以在加载失败/初始化时显示默认的图片。需要了解Volley图片加载机制，可以去看Volley源码中的toolbox::ImageLoader类
2. 要监听、处理GalleryView的滚动事件，我们先要理解HorizontalScrollView（以下简称HSV）的滚动距离（GalleryView继承自HSV）。见下图：

假设我们的屏幕是上图中的绿框，而上图的蓝框是我们的HSV,则此时HSV的滚动距离(scrollX)为上图红线所表示的距离。而黄线则为HSV的宽度

**实现：**重写onTouchEvent()方法。当动作为MotionEvent.ACTION\_MOVE时，如果滚动距离scrollX>=mChildWidth（图片宽度）时加载后一张图片，如果scrollX==0,则表示当前滑到GalleryView的最左，则加载前一张图片。**需要注意的是：**在GalleryView的宽度小于屏幕宽度时，或者GalleryView宽度<屏幕宽度+mChildWidth时（当图片总数小于X时），是无法触发上述的加载后一张图片的条件的。因此需要定义第二种加载方式，即同时存在两种加载方式——在ACTION\_DOWN时记录x坐标downX,在ACTION\_UP时再次记录x坐标upX，计算downX与upX之差就是滚动的距离。即可根据滚动距离触发图片的加载。

**保证每次只加载<=X张图片：**在加载后一张图片时，移出当前GalleryView中的（其实是LinearLayout中的）第一张图片，并插入新图片到尾部；在加载前一张图片时，移除最后一张图片，并插入图片至头部

1. 设置图片(Item)点击事件，实现点击接口回调，定义一个Item点击监听接口OnItemClickListener，接口中定义了一个回调方法onItemClick()。在GalleryView中为每一个ImageView添加点击事件:

imageView.setOnClickListener(listener);并在listener#onClick()方法中调用mItemClickListener#onItemClick()从而实现接口回调。需使用一个Map<View,int>存放ImageView与position的对应关系，在onClick()方法中调用onItemClick()方法中才能将正确的position传递给监听者。

**实现点击时滚动到相应位置:**理解了GalleryView的滚动距离，实现这个功能也很容易——只需要判断被点击的图片是GalleryView中的第几项，然后调用scrollTo()/scrollBy()滑动相应的距离即可。然而**需要注意的是：**由于我们使用的是动态加载的方法，即滑动加载后一张/前一张。而scrollTo/scrollBy方法是不会触发onTouchEvent()方法的，所以我们得手动实现向前/向后加载N张图片，不然是无法滚动到相应位置的。

说得比较抽象，拿个例子来说就是，如果你点击第3张图片，想要将这张图片滚动到第1张。则需要向后加载2张图片，再将GalleryView滚动到(0,0)的位置，因为**向后加载时，也在向前销毁**。就是说点击的图片此时已经成为了GalleryView的第一张图片了。所以只需滚动到GalleryView的最左的位置即可。

1. 定义加载更多接口。类似于定义图片点击监听接口。在加载后一张图片/前一张图片的方法中，当已经加载到最后一张图片时(mLastIndex == mSize-1)，判断当前GalleryView是否有设置了加载更多接口，如果有则调用加载更多的方法onLeftLoadingMore()/onRightLoadingMore(),如果没有则循环加载图片。
2. 一些方法解释：
3. setAutoLoad(boolean autoLoad)——设置点击监听时是否需要调用loadNext()、loadPre()
4. setCurrentSelectItem(boolean isByLyric,int index)——设置当前选中的子项（图片）,isByLyric传递给onItemClick()方法，用于区分是否需要设置音频联动。
5. initData(GalleryAdapter adapter)——初始化GalleryView的数据，获取X（最大同时加载图片数），初始化adapter并且加载第一屏图片
6. updateIndex(int count)——更新Map<View,int>中各个View对应的position,因为从首部插入view，所以都需要改变。
7. getIndexOfViews(View view)——获取传入view是GalleryView中的第几个view。
8. destroy()——回收所有的图片

##### 设置各部分联动

1. 设置歌词播放、显示与ProgressBar联动
2. 设置歌词播放、显示与图片展示联动

以上两点都是由音频播放进度改变导致歌词切换显示，进而影响ProgressBar的进度改变和图片切换，因此可以在LyricView中定义一个线程用于作为一个监听器，定义一个进度监听接口实现与外界进行交互，每隔一段时间调用一次该接口的回调方法，（相当于给实现者传递一次信息）该信息中包含当前音频的播放进度。

随行导游界面通过给LyricView对象设置进度监听接口，就可以实现歌词播放与ProgressBar和图片展示的联动。通过以下方法可设置

ProgressBar#setProgress(int)

GalleryView#setCurrentSelectedItem(boolean, int)

1. 设置图片点击与歌词播放联动

由于有两个GalleryView,一个是展品的多角度图片展示，一个是展品列表的图片展示。两个需要实现的点击效果是不同的。

1. 多角度图片联动

点击多角度图片GalleryView中的图片时，应该切换当前播放进度，由

LyricView#setCurrentPosition(int)可设置

1. 展品列表联动

而点击展品列表GalleryView中的图片时，应该切换整个页面数据，因此要重新获取数据，并刷新LyricView、多角度图片和展品列表的GalleryView。

#### 专题导游界面

该界面主要由3部分组成——头部+标签栏+筛选的展品列表（上拉加载ListView，见上文）

主要实现难点在于标签栏的实现以及ListView悬浮头部的实现。

该界面整体结构实质是一个ListView（展品列表）+头部（标签栏），这样设计的好处是使整个页面均可滚动，且对于实现一些比较好的UI效果比较方便，比如实现悬浮头部。后期可考虑将该页面的实现方式换成RecyclerView。

##### 标签栏的实现

分析一下，可知一个标签栏包含了一个标签名，和若干个标签项，如图：



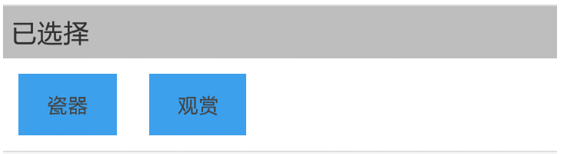
由于实际中标签栏可能会有多个，因此我们将上图该结构的View设置成自定义View——LabelView,以提高代码复用度。

**LabelView，自定义标签栏View的实现**

分析一下该View,可知该View的组成为一个TextView+一个GridView。这个GridView比较特殊，因为可能有很多个，子项总宽度超出了屏幕宽度，因此必须支持水平滑动，所以应在外层嵌套一个HSV（HorizontalScrollView）。而GridView的子项其实就是一个Button（可自定义样式）。

该View的实现难点在于如何设置子项的点击事件监听，由于我们标签栏中有一个已选择栏，用来显示当前被选中的标签。则点击监听需要完成两项工作：

1. 子项被点击时，要改变其背景，如上图，背景色变灰表示被选中
2. 子项被点击时，需要将该项添加到已选择标签栏中。



此外，由于我们认为已选择标签栏实质也是一个标签栏，即应适用LabelView，因此我们认为标签栏的点击事件还应支持：

1. 已选择标签栏点击时，对应子项的背景色变回未选择的背景色，且该项在已选择标签栏中移出。

也就是说事实上，点击事件的实现是由具体的LabelView实现决定的，因此应该封装点击监听接口，并由实现者去实现监听器应实现的逻辑细节。

具体实现细节参见.widget.LabelView类和 .adapter.GridAdapter类及 .ui.SubjectSelectFragment类

##### ListView悬浮头部

ListView悬浮头部有一种十分简单的设计思想，就是采用“分身”——就是拷贝一个要悬浮的View的副本(原View称为originView,副本称为invisView),**注意：invisView应该与ListView放在同一个FrameLayout下,并且设置不可见。**

<FrameLayout

android:id=*"@+id/frag\_subject\_select\_invis\_layout"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"wrap\_content"*>

**<invisView …/>**

<com.app.guide.widget.AutoLoadListView

android:id=*"@+id/frag\_subject\_lv\_exhibits"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"wrap\_content"*/>

</FrameLayout>

然后计算该originView在ListView中的位置，比如originView是ListView的第二个头部，那么它的位置就是1(从0开始排序)。之后监听ListView的滚动状态：

ListView#setOnScrollListener(new ScrollListener(){…});

在onScroll(…,int firstVisibleItem,…)方法实现中进行判断，如果当前已经滚动到originView的位置或以下(如firstVisibleItem>=1时)，将invisView设置为可见，而当firstVisibleItem<1时，设置invisLayout为不可见。这样看起来好像就是originView悬浮了一样，其实显示的是它的“分身”，只是我们看不出来而已。这样就实现了悬浮效果。

如不清楚可参考博客：

http://blog.csdn.net/u011717684/article/details/45296627

#### 地图导游界面

该界面主要是用SurfaceView绘制图片，以及控制图片的缩放、拖拽等。

##### 自定义地图类——MyMap（继承自SurfaceView类）

1. 处理缩放和拖拽事件

在这里我利用了Matrix类提供的图片操作方法去进行图片的缩放和平移处理，关于该方面的知识可以参考

<http://blog.csdn.net/smbroe/article/details/43792617>

1. 双击放大

为了实现双击放大，在这里我们MyMap类中设置了一个成员变量lastClickTime用来记录上一次点击屏幕的时间（点击屏幕的时间值可以通过 MotionEvent的getEventTime方法去获得，单位是ms），如果当前点击事件的时间与上次点击事件的时间差值小于300ms则执行放大事件。

1. 多点触摸放大

通过MotionEvent中的方法来获得两个触摸点之间的距离大小， 如下：

//计算两个触摸点的距离

private float spacing(MotionEvent event) { float x = event.getX(0) - event.getX(1); float y = event.getY(0) - event.getY(1); return (float) Math.sqrt(x \* x + y \* y);}

利用一个变量oldDist表示前一次两个触摸点的距离，利用一个oldRate表示前一次的缩放，在onTouchEvent方法中move的情况下不断更新当前缩放mCurrentScale = oldRate \* (newDist / oldDist);

1. 地图拖拽

利用一个PointF变量mapCenter表示当前地图中心的位置在手机屏幕上的坐标，当拖拽事件发生时通过手指移动的距离来不同更新mapCenter的值，并在draw方法中利用Matrix类操作图片

matrix.postTranslate(mapCenter.x - mBitmap.getWidth() / 2, mapCenter.y - mBitmap.getHeight() / 2);

1. 添加地图标记

编写一个MarkObject类来表示地图标记，在该类之下存储了标记的Bitmap对象，该标记相对于整张地图的位置，以及点击标记的回调事件的处理。

在MyMap类中利用一个List变量markList来记录所有已经添加的地图标记。

1. 处理标记随拖拽和缩放事件而改变位置：这里主要是根据mapCenter的点来进行计算，具体的计算可以参考代码。
2. 处理点击事件：在onTouchEvent方法中up情况时，遍历markList中的MarkObject进行判断当前触摸点是否被包含在当前的标记区域中；
3. 异步线程绘图方法

首先在构造方法中开启一个新的DrawThread处理绘图事件，之后当我们每一次需要重写绘图时通过Handler传递绘图消息，接着在DrawThread去处理该消息，调用绘图方法进行异步绘图。

关于自定义地图可以参考

http://blog.csdn.net/smbroe/article/details/44520881

1. 方法解释
2. setBitmap(Bitmap bitmap)方法--设置地图显示的Bitmap；
3. setLoactionPosition(float x, float y)，设置定位到当前的地图坐标；
4. addMark(MarkObject object)--为地图添加一个标记
5. zoomIn()--将地图放大一个级别
6. zoomOut()--将地图缩小一个级别
7. adjust(float px, float py)，将地图中心自适应的移动到地图坐标(px, py)

### 侧边栏界面

侧边栏使用了SlidingMenu开源库//TODO 地址

侧边栏的界面组成实质就是一个列表（ListView），其item中包含一个ImageView(菜单项的图标)和一个TextView(菜单项的标题)。

主要就是实现一个Adapter，用于菜单项的数据加载。

所有包含TopBar的界面都包含侧边栏，即能调出侧边栏。

侧边栏可由两种方式调出：

1. 通过点击TopBar的ivMenu（菜单按钮）。
2. 通过从左往右滑动

由于侧边栏的滑动调出与各界面子控件的滑动发生冲突，经常会出现误滑的现象，所以目前将侧边栏的侧滑弹出功能取消掉，只能通过TopBar点击触发。//TODO

### 下载中心界面

下载中心界面模仿百度地图离线中心界面。

数据来源：由实时API获取

页面构成：//TODO

### 搜索界面

搜索界面主要分为两个部分：SearchView（自定义搜索View）+搜索结果列表(ListView)。

#### 自定义SearchView的实现

自定义SearchView应该包含两个结构：输入框+弹出框。弹出框包含两个功能——用于自动补全和热门搜索推荐，所以应分为两个列表：自动补全列表和热门搜索列表

分析该View应具备的主要功能有：

1. 提示框的数据与输入框输入的文本是实时联动的。
2. 应保证与结果列表的联动性，本程序中采用接口回调的方法实现。SearchView捕捉搜索动作后（用户点击软键盘的搜索键），调用定义的搜索回调接口的方法，将搜索指令传递到实现者，使得实现者可以实时更新结果列表。
3. 同理应保证与自动补全列表的联动
4. 输入框的UI应是动态的，即UI随着输入的文本的改变而改变，如：在未输入文本时，清除按钮这里写图片描述应该是隐藏的；只有当框中有文本时才会显示。 当输入框有文本时，点击输入框，应显示自动补全列表框。
5. 软键盘也应该是动态的，如完成搜索时应自动隐藏。
6. 选择提示框的选项会自动补全输入框，且自动进行搜索

根据以上分析，我们需要

1. 给SearchView的输入框（EditText）设置文本改变监听器editText.addTextChangedListener(new TextWatch(){…})
2. 给输入框设置点击监听
3. 给提示框设置点击监听
4. 定义一个回调接口，定义2个回调方法：

onSearch(String text):完成与结果列表的联动

onAutoRefreshComplete(String text)：完成与自动补全列表的联动

基本上以上4点完成之后，我们的SearchView也就实现完成了。

如果不清楚可以参考博客：

http://blog.csdn.net/u011717684/article/details/46510241

### 设置界面

设置界面采用PreferenceFragment实现，通过编写preference.xml文件，让系统自动生成界面。其中preference.xml应放在项目的xml文件夹下。文件中一个<PreferenceCategory>标签代表一个设置栏目：如个人设置、系统设置等。<PreferenceCategory>标签下有多种标签，分别代表不同的设置条目，目前我们只用到<CheckBoxPreference>带有CheckBox的设置条目，和<PreferenceScreen>普通条目。各个设置条目都有一个key属性，通过该key属性，就可以在代码中取得各个设置项，并监听每一项的点击事件。

### 对话框

使用SweetAlertDialog这个开源库。

项目地址为

https://github.com/pedant/sweet-alert-dialog

自定义多两种对话框：带CheckBox对话框 和 专门用以显示消息的对话框