中山大学移动信息工程学院本科生实验报告

年级	1501	专业（方向）	移动（互联网）
学号	15352015	姓名	曹广杰
电话	13727022190	Email	1553118845@qq.com
所在小组	58	负责	主题切换与自定义组件

Content

一、实验题目

期末Project第58组项目——日迹

二、实现内容

本次实验模拟实现一个便签记录。

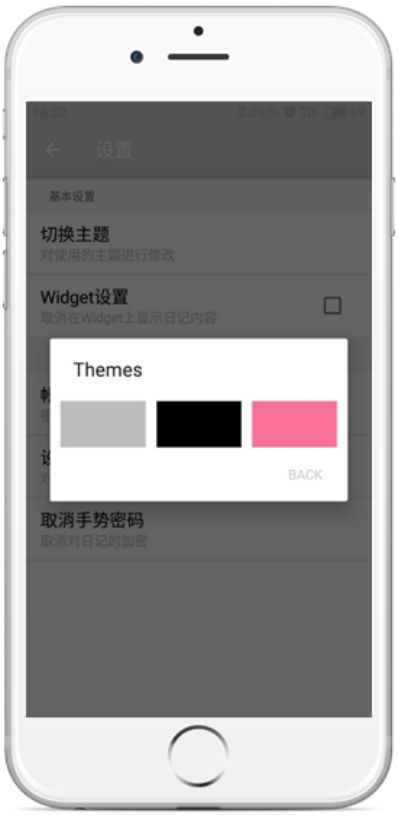
- 主要功能，日记的编写与添加；

- 日历功能，根据关键字查询日记；
- 用户的注册以及登录；
- APP的主题切换功能；
- 日记的桌面widget小部件；
- 手势加密；

笔者负责APP的**主题切换部分**以及**日历部分的自定义部件**的使用。

三、课堂实验结果

实验截图



实验步骤以及关键代码

笔者负责实现主题换肤的实现，此外负责查找和分析自定义控件的使用以便确定自定义控件的使用，为备忘录界面提供接口使用方法。

自定义控件的继承

自定义控件的实现，由于要使用自定义控件实现日历的设计，自定义的控件首先需要继承于view的class；

```
public class BtCalendarView
    extends LinearLayout implements OnDateSelectedListener{
      /* Implement */
}
```

其中，`LinearLayout`继承于`ViewGroup`，而`ViewGroup`继承于`View`，迭代的继承实现使得当前笔者自定义的控件也可以有控件的效果。

确定控件的时间管理接口

笔者的控件用于实现一个日历，所以在控件的内部需要有日期的管理者。其中时间管理接口包括月份管理、年份管理以及日期管理。这里以月份管理为例，说明如何使用控件中的月份的点击以及跳转。

```
public abstract class BtMonthViewProvider {
    /* set and get function. */
    public abstract View getView();
    public abstract void updateView();
    public void setOnDateSelectedListener(OnDateSelectedListener listener){
        mListener = listener;
    }
}
```

1. 数据管理类

   月份信息的管理类哟用于管理数据，便于月份管理的信息：

   ```
   public class BtMonth {
       private int mMonth;
       private int mYear;
   }
   ```

   内部会有一些set函数、get函数以及一些比较函数，都比较常规，此处就不予赘言了。

2. 抽象函数

   从以上的代码框架可以看出，在该实现中有两个抽象函数，即`abstract`函数，声明为抽象的函数是因为期望该函数在不同的场合可以有不同的具体的作用。函数`getView`用于返回月份的信息，函数`updateView`用于更新当前的展示界面——但是具体使用什么view则是要根据使用的场合有不同的使用方法。

   而在本次实验中，虚函数的使用并不是因为在本次实验中有重载的操作，本次实验中使用的依然是这两个函数原本的作用。但是虚函数的声明会为后续开发的继承带来方便，因为当调用函数的环境复杂些时，就可能需要对这两个函数进行重载。

3. 接口信息

   接口信息的使用主要是作用在`setOnDateSelectedListener`函数中，该函数主要用于记录某一个日期是否被点击了。至于点击之后具体进行怎样的操作，也是需要在具体的场合进行具体的实现。

接口`interface`的使用

接口的使用与class的使用非常类似，这一点从二者的声明上就可以看出：

```
// declaration of interface ;
public interface Interf{
    public void Inter();
}

// declaration of class
public class Clas{
    private String s;

    private void Func(){
        /* Implement. */
    }
}
```

接口在声明之后，使用时不必实例化，但是同时这也就意味着接口在实现的时候不允许使用内部的变量。与之相对的，class在声明的时候需要实例化。

接口使用时候的结构如下：

```
// implement of a class calling interface
public interface A{
    private Interf interf;
    public void setInterf(Interf interf){
        this.interf = interf;
    }
}

// actually use in the main activity;
public void MainActivity extends AppCompat{
    private A a;
    public void func(){
      a.setInterf(new Interf(){
        /* implement. */
      })
    }
}
```

其中A可以是任何一个调用该接口的类，只需要在内部实现set接口的函数即可。

日历的表格结构

在数据管理的部分实现之后，针对展示的部分，按照习惯，日历信息需要展示为表格的形式。为此需要针对表格的需求，实现一个简单的有表格格式的日历。本来对此的实现是可以通过`RecyclerView`的，但是`RecyclerView`的许多实现暂时都无法用到，这里只需要较为简单的功能，于是使用了继承`GridView`的策略。

1. `GridView`的使用方法

```
private GridView mGridView;
public View getView() {
    // initialization
    // 7 days a week
    mGridView.setNumColumns(7);
    /* other implement. */
    return mGridView;
}
```

最为重要的函数是`setNumColumns`，在本次实验中，该函数的使用是由于日历的信息有7列，表征一个星期有七天。通过该函数的额使用，可以在屏幕上构建出一个上简单的表格结构。

2. `GridView`所在界面

当前仅仅是完成了对于表格的设计，但是表格究竟在何处调用还尚未确定。所以在设置`Grid`的时候，需要一个参数用于获取`GridView`是在哪一个布局文件中，这里使用到的参数就是`Context`。

由于该部分期望表格的初始化在自定义的类内部完成，而不是在调用的位置完成。这里就需要context来确定究竟是哪一个activity调用了该界面：

```
    public View getView() {
        // Creates the grid view
        mGridView =
            (GridView)LayoutInflater.from(mContext)
            .inflate(R.layout.grid_view, null);
         return mGridView;
    }
```

`.from`之后的参数用于表征当前处于哪一个界面中（一个界面对应一个布局文件），此后的inflate函数将会将其中的`GridView`找出用于初始化。

主题换肤的实现[1]

1. 自定义主题信息

实现主题换肤需要首先自定义主题信息，自定义主题常规情况下需要在value文件夹内部的`style.xml`文件中添加主题设置：

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
</style>
```

在这个主题中，对于颜色信息进行了定义，因此，其他activity中所有使用到当前主题颜色的界面锁展示的颜色都是该主题下所定义的颜色。由于这是原始的主题信息，笔者在原有的基础上定义一个新的主题，二者的结构是相似的：

```
    <style name="AppDark" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/black</item>
        <item name="colorPrimaryDark">@color/black</item>
        <item name="colorAccent">@color/black</item>
    </style>
```

在笔者新定义的主题中，将使用到的颜色都设置为了黑色，因此在对于activity换肤的时候，所有使用该主题的界面都会变得神秘而深沉，简而言之就是所有看到的颜色都是黑色了，其中color中的black需要自己定义。

与之类似的，还可以定义很多类似的主题，用于界面视图的切换。

2. 主题管理类

上文介绍了笔者自定义主题的操作，本节介绍如何为activity设置主题。设置主题的主要函数是`setTheme`，但是由于该函数需要作用于多个activity，这里笔者添加了一个管理类实现对于主题的管理：

```
public class ThemeUtils{
  // An index to destinguish different theme
    private static int sTheme;
  // set theme
    public static void setmTheme(int theme){
        sTheme = theme;
    }
  // acomplish theme into an activity
    public static void onActivityCreateSetTheme(Activity activity){}
}
```

笔者在该类使用一个静态的整数值记录期望使用的主题，而表示主题的参数将会通过接口`setmTheme`传入。最后通过函数接口`onActivityCreateSetTheme`在不同的activity中使用，实现主题颜色的切换。

函数`onActivityCreateSetTheme`：

```
    public static void onActivityCreateSetTheme(Activity activity){
        switch (sTheme){
            default:
            case 0:
                activity.setTheme(R.style.AppTheme);
                break;
            /* other theme. */
        }
    }
```

在该函数中使用了函数`setTheme`对作为参数传入的activity进行主题设置，需要注意的是，主题的设置应该放在布局文件加载之前，通过`OnCreate`函数运行该接口，实现主题的设置。因为主题的正式加载还是在布局文件的加载中实现的，先加载了布局文件，屏幕上所展示的控件以及信息就已经确定了，再设置主题信息是没有效果的。

使用样例：

```
public class SettingActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
         // practical use of ThemeUtils
        ThemeUtils.onActivityCreateSetTheme(this);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_setting);
        /* Other Implement in onCreate. */
    }
}
```

3. activity的刷新

由于设置主题的操作是在`onCreate`函数内部，因此只有在初始化一个activity的时候，主题设置才会真正起效。同时这也意味着，在设置主题的界面使用结束后，设置界面是不会及时修改主题的。除非关闭当前界面再重新打开，但是这样就会导致当前界面的闪烁，造成不良的用户体验，因此笔者利用界面的跳转信息实现所有activity的刷新：

```
@Override
public void onClick(View v) {
    ThemeUtils.setmTheme(1);
    getActivity().finish();
    Intent intent = new Intent(getActivity(), MainActivity.class);
    startActivity(intent);
}
```

这里笔者是使用`AlertDialog`中的Button进行触发，从当前界面直接跳转到`MainActivity`就会触发`MainActivity`中的`OnCreate`函数，于是`MainActivity`实现初始化，主题信息得以设置。同时当前设置界面被终止，再次跳转到当前界面的时候会触发该界面的`OnCreate`函数。

自定义对话框

在选择主题的具体实现中，笔者使用的是自定义的对话框。自定义对话框用于设置对话框中的具体布局。

```
View theme_view = factor.inflate(R.layout.theme_option, null);
/* set theme_view... */
@Override
public boolean onPreferenceClick(Preference preference) {
    AlertDialog.Builder chglog = new AlertDialog.Builder(getActivity());
    chglog.setTitle("Themes")
          // set view
          .setView(theme_view)
          .show();
    return true;
}
```

设置view信息的时候，笔者使用了一个view对来自于对应布局文件中的布局信息进行拓展，该布局正是笔者期望在对话框中展示的布局，之后对该view进行一定的设置，此时view已经实现了对于期望布局的设置，将其作为参数通过函数`setView`传入对话框中，就可以得到期望的自定义对话框了。

实验遇到困难以及解决思路

1. 从`MainActivity`跳转到`Fragment`会闪退

   现象：

   - debug的时候尚未进入`Fragment`的初始化函数

   由于尚未进入`Fragment`中，所以问题并非是在`Fragment`中，经过debug查询得知在`MainActivity`中有一个用于跳转的变量没有被初始化，始终为`null`。该问题通过debug的逐行检测可以查出。事实上，闪退信息出现之后几乎都可以通过debug进行排查。

2. `activity`累积未被关闭[2]

   现象：

   - 点击向前跳转正常
   - 点击向后跳转，发现此前出现的所有界面都未被关闭
   - 界面交错，不宜在其他的`Activity`中使用finish关闭跳转的来源，记录操作复杂

   笔者最后使用`setFlag`对跳转的intent进行设置，清空`Activity`栈：

   ```
   @Override
   public void onClick(View v) {
       ThemeUtils.setmTheme(1);
       Intent intent = new Intent(getActivity(), MainActivity.class);
       // clear activity stack
       intent.setFlags(
           Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK
       );
       startActivity(intent);
   }
   ```

3. 无法使用`Get`获取`Json`数据

   现象：

   - Get的网址有参数传入
   - 仅使用Get会导致闪退

   之前涉及到网络请求的时候，需要使用Get获取网络中的`Json`数据，但是貌似实验中的使用方法并不足以应对需要参数传入的API，因此笔者使用了`Query`实现参数的传入。

4. 使用`Query`报错`Http403Forbidden`

   现象：

   - 可以编译运行
   - 回调函数显示`onError`
   - log的信息为`Http403Forbidden`

   经过查询得知，该报错信息意为发送信息的格式正确，服务器也可以理解，但是由于权限问题不予回应。

   解决方案：笔者换了另一个服务器的API就可以使用了，这说明不同的服务器对于权限的要求是不同的。

5. log报错信息`Expected BEGIN_OBJECT but was BEGIN_ARRAY`

   现象：

   - 程序可以编译、运行
   - 信息可以发送，服务器返回值
   - 进入`onError`的回调函数

经过查询得知这种情况的原因是服务器的返回值不规范，有时候为数组格式有时候为Object格式，需要使用对应的Model对其进行储存。

6. 跳转到`MainActivity`会闪退

首先使用debug针对运行的函数栈进行排查，排查的结果显示已经进入`MainActivity`的`OnCreate`函数，在`OnCreate`函数中某处闪退。排查的结果显示，该界面中有用于处理跳转来源的字符串，但是该字符串没有初始化赋值，在赋予其`null`的时候，由于内存空间出错闪退。

解决方案：将字符串初始化为任意值，并且在为字符串赋值之前检查右值是否为空。

7. 接口的使用

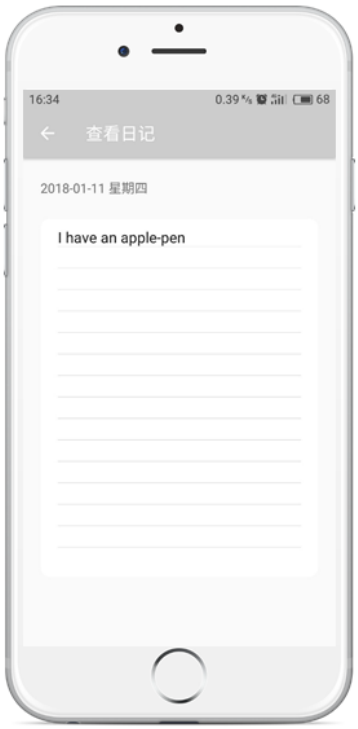
接口的使用架构不明。

笔者在分析自定义控件的时候，需要使用接口，但是接口的使用方法与常规的class使用方法有所不同，笔者花费了一些时间查询到接口的使用方法。[3]

四、课后实验结果

以下为展示APP的两种主题，粉红色与银白色：

red theme



grey theme

五、实验思考及感想

本次实验中笔者遇到了很多问题，解决问题的效率也很低，总之整个学期笔者都是在android的“控件不知道该怎么用”与“这个bug是什么意思啊”中辗转反复，当然也不是一点进步没有。之前的很多问题都是笔者的代码风格的问题或者是有一些问题没有考虑到，而这一次整合的project中，笔者遇到的问题其实是真的有一点匪夷所思。

在实现对话框的时候，为对话框添加自定义布局，就会有很多问题，这里笔者列举一个例子：

```
1  void Init(){
2      // initialze user defined view;
3      View view_in = factor.inflate(R.layout.dialog_update, null);
```

```
}

adapter.setOnItemClickListener(new CommonAdapter.OnItemClickListener() {
  @Override
  public void onClick(final int position) {
      // intialization;
      final AlertDialog.Builder updatelog
                          = new AlertDialog.Builder(MainActivity.this);
      // Begin set Dialog
      updatelog.setView(view_in)
                .setPositiveButton("Yes",
                                   new DialogInterface.OnClickListener() {
                     @Override
                     public void onClick(DialogInterface dialog,int w) {
                       // Action in the click;
                     }
                })
                .show();
  }
```

这种循环的点击监听设置是非常常见的，问题出现在`Begin set Dialog`部分，笔者为对话框设置自定义布局的时候，第一次触发是没有问题的，但是在第二次触发的时候就会崩溃——而且并非运行到某一条语句之后崩溃，而是运行完最外层的click监听之后崩溃。笔者花费了很多的时间解决这个问题（因为不知道问题出在哪一条语句），最后解决了这个bug。

现象：

在第一次调用的时候view刚刚被初始化完，此时的view是包含信息的，然而在第二次调用的时候，view的值就变为null了——可是这是全局变量，自始至终没有更改过，是系统自己对该信息进行了修改。

问题原因：

java会对长期没有使用过的变量进行优化（有时候是用户不期望出现的优化），在上文的情况下就是java对笔者的view变量进行了优化，导致setview的时候传入的参数值是不可查变量，在函数运行完的时候，这种情况造成了非常严重的结果，于是程序崩溃。

这一次不是笔者使用的代码的问题，而是整个语言体系的问题，如果在一份几百行的项目找那个都会由于语言结构出现这种潜在性的问题，那么在一个大的项目中究竟会有多少尚未察觉的内存使用不当的问题。这种情况就已经是在说明，当前的Android体系架构是有很懂潜在的隐患的，而且由于针对java的开发包没有一个具体的管理标准，在运行的过程中总会有一些潜在的问题，比如对话框不能添加侧边栏——尽管可能会有一些奇怪（但是逻辑上来说，布局文件使用drawer布局是完全可行的，可是就是会出现一些闪退的问题）更何况java这个语言还并不安全。所以一场改革真的是势在必行。

就是因为这些不成体系的架构，笔者的Android学习过程总是不太愉快，接口设计的并不算简洁方便，函数的使用也不是非常明了。最让笔者感到不能忍受的是组件的使用，笔者认为组件的使用应该有一个统一简单的架构，而不是每一个组件有自己独立的接口和使用位置（另外，接口的使用需要有高度封装的特点）。

在落实到具体的bug排查的时候，最可怕的就是闪退的问题。闪退的问题大多都是同一个原因，即内存的错误。一般有两种现象：

1. 一种是在某一条语句中出现了内存的错误使用；
2. 一种是变量由于java的语言体系结构被优化掉了；

对于这种情况，可怕的原因就是开发环境不能找到问题所在，在运行设备上的程序退出之后，整个开发环境运行的程序也直接退出，不会留下报错信息。因此对于此类问题，最实用的方法就是使用添加断点的方法，**逐步缩小排查范围**，最后确定是哪一条语句或者是哪一条设置的变量出了内存问题。

笔者在使用网络管理的语句与网络交互的时候，一开始是query的使用方式不对，但是在query的使用方式修改过来之后，服务器的报错信息是403。嗯这是说服务器理解当前客户端的申请权限，但是就是不给反馈数据，因为不能确定使用者的身份，这种设计多半是处于安全考虑，但是笔者又不知道如何确定一个手机的头部和身份信息……添加头部信息之后，后端返回的JSon数据又不能被解析——原来返回的数据有时候是对象类型的，有时候是数组类型的……后端的返回数据不成体系结果前端的设计实现跟着遭殃，安卓的开发实在是太随意了，这种开发体系不会长久的。至少从笔者的观念中，这些实际的实现设计，尽管不能说是完美无缺，但是基本的交互体系结构还是要遵守的，毫无体系的开发架构必然会导致无谓的工作量的增加，浪费开发者的时间与精力，这是非常愚蠢的行为。

---

1. 摘自edwardsbean的博客http://blog.csdn.net/wsscy2004/article/details/7562909↩

2. 摘自Crazy_弑神风的博客http://blog.csdn.net/wsscy2004/article/details/7562909↩

3. 摘自余小涛的博客http://blog.csdn.net/qq_23940659/article/details/50791721↩