**AndroidOSettings源码流程分析（静态界面篇）**

置顶2018年10月19日19:07:09[又双叒叕Fixed了](https://me.csdn.net/qq_34149526)阅读数：231

**AndroidOSettings**

* [静态界面篇](https://blog.csdn.net/qq_34149526/article/details/83186719)
* [界面渲染篇](https://blog.csdn.net/qq_34149526/article/details/83211665)
* [数据加载篇之一级菜单](https://blog.csdn.net/qq_34149526/article/details/83239567)
* [数据加载篇之二级菜单](https://blog.csdn.net/qq_34149526/article/details/83278597)
* [数据加载篇之获取及修改默认设置属性值](https://blog.csdn.net/qq_34149526/article/details/83307695)
* [搜索栏篇](https://blog.csdn.net/qq_34149526/article/details/83341195)

Settings

***路径：packages/apps/Settings***

**主[界面](https://www.baidu.com/s?wd=%E7%95%8C%E9%9D%A2&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)和子界面**

从清单文件AndroidManifest.xml中入手：

setting的主界面的Activity为Settings

Settings.java类看，除了大量静态类继承SettingsActivity，就没什么东西了。

setting的子界面Activity为SubSettings

SubSttingsjava里有重写了两个方法向上导航和判断是否有效fragment，还有一个静态内部类继承本身。

Settings与SubSttings都是空Activity（不继承7大生命周期）

Settings与SubSttings都继承于SettingsActivity

SettingsActivity继承自SettingsDrawerActivity,而SettingsDrawerActivity是在SettingLib中定义，SettingsDrawerActivity名称是沿用N版本的名字,但是侧滑栏功能在O版本上已经移除了。

主要分析的是SettingsActivity的**onCreate**()方法；

布局加载：

主界面的layout为：settings\_main\_dashboard.xml:

主界面布局中包含两个FrameLayout，一个是顶部搜索栏，还有下面的主要内容（显示一级菜单）

参考7.0版本主界面布局：

子界面的layout为：settings\_main\_perfs.xml:

子界面由一个LinearLayout和一个RelativeLayout组成。RelativeLayout默认是隐藏的。

界面如下：

Settings界面：

主界面使用DashboardSummary(Fragment)进行填充

子界面则使用各自的Fragment,并设置各自的主题。

# AndroidOSettings源码流程分析（界面渲染篇）

2018年10月20日11:16:07[又双叒叕Fixed了](https://me.csdn.net/qq_34149526)阅读数：342

**AndroidOSettings**

* [静态界面篇](https://blog.csdn.net/qq_34149526/article/details/83186719)
* [界面渲染篇](https://blog.csdn.net/qq_34149526/article/details/83211665)
* [数据加载篇之一级菜单](https://blog.csdn.net/qq_34149526/article/details/83239567)
* [数据加载篇之二级菜单](https://blog.csdn.net/qq_34149526/article/details/83278597)
* [数据加载篇之获取及修改默认设置属性值](https://blog.csdn.net/qq_34149526/article/details/83307695)
* [搜索栏篇](https://blog.csdn.net/qq_34149526/article/details/83341195)

**Settings渲染fragment（菜单之间的跳转）**

在Activity中填充fragment的方法主要为SettingActivity中的**SwitchToFragment**()

给mFragmentClass赋值为AndroidManifest.xml中的<meta-data>标签中key为**META\_DATA\_KEY\_FRAGMENT\_CLASS**的的值

**getMetaData**()方法

代码如下：

这个函数的主要作用就是从Activity标签中获取meta-data标签中key为com.android.settings.FRAGMENT\_CLASS的值，并将其赋值给**mFragmentClass**这个私有变量。

构建一个intent并为其增加一个特殊的键值对

key：settings:show\_fragment(EXTRA\_SHOW\_FRAGMENT)，

value：mfragmentClass指定的类名

**所以要先执行getMetaData，是因为mFragmentClass赋值是在getMeatData中进行的。**

判断布局

判断当前类是否为Settings.class或者是其他子菜单，即进入方式为点击launcher上的图标进入的主设置[界面](https://www.baidu.com/s?wd=%E7%95%8C%E9%9D%A2&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)

接下来看**launchSettingFragment**()方法：

initialArguments通过赋值保存了meta-data中指定的value值。

在SwicthToFragment实现渲染

**加载DashboardSummary(Fragment)显示一级菜单页面；**

**其他的则加载子菜单页面；**

**switchToFragment**()方法：

先判断fragment是否合法，然后将方法中传入的Fragment进行实例化，通过**replace**(fragment)进行渲染。

**通过上面的分析，我们应该了解的事情是，AndroidManifest中每个Activity其meta-data中的数据都是很有用的，特别是com.android.settings.FRAGMENT\_CLASS对应的键值对，指定了其所在Activity的真正布局实现。其实，也是完成了从其他包的Activity向Settings中fragment的跳转实现。**

[***点击进入下篇——数据加载篇***](https://blog.csdn.net/qq_34149526/article/details/83239567)

**AndroidOSettings源码流程分析（数据加载之一级菜单）**

2018年10月21日10:36:36[又双叒叕Fixed了](https://me.csdn.net/qq_34149526)阅读数：515

**AndroidOSettings**

* [静态界面篇](https://blog.csdn.net/qq_34149526/article/details/83186719)
* [界面渲染篇](https://blog.csdn.net/qq_34149526/article/details/83211665)
* [数据加载篇之一级菜单](https://blog.csdn.net/qq_34149526/article/details/83239567)
* [数据加载篇之二级菜单](https://blog.csdn.net/qq_34149526/article/details/83278597)
* [数据加载篇之获取及修改默认设置属性值](https://blog.csdn.net/qq_34149526/article/details/83307695)
* [搜索栏篇](https://blog.csdn.net/qq_34149526/article/details/83341195)

**数据加载之一级菜单**

**DashboardSummary**是顶级菜单的容器,那么数据的获取和它也就有关系。

DashboardCategory的获取是在DashboardSummary中的**updateCategoryAndSuggestion**方法中，获取之后在**DashboardAdapter**设置

根据"com.android.settings.category"的值查询子项数据，这里的值为"***com.android.settings.category.ia.homepage***"。

在DashboardSummary的onCreate函数中有获取的有两个很重要数:**mDashboardFeatureProvider**,**mSuggestionFeatureProvider**.这两个是主要的数据提供者,mSuggestionFeatureProvider和mDashboardFeatureProvider的数据获取是有所不同的,重点说明下**mDashboardFeatureProvider**.

**mDashboardFeatureProvider**提供的数据是一级菜单如"电池","显示","网络和互联网"等,

实现类为**DashboardFeatureProviderImpl**.java,而DashboardFeatureProviderImpl中的具体的数据是通过函数**getTilesForCategory**()从CategoryManager获取的.

**CategoryManager**是SettingsLib这个静态包中公共类,可以看一下:

可以看到CategoryManager是一个单例类型,这里就是真正的数据加载位置,

加载是通过函数**reloadAllCategories**()调用**tryInitCategories**()获取的.

到这里为止,整个数据获取的流程已经清楚,但是数据加载是在哪里完成的呢,还要回到**SettingsDrawerActivity**中:

因为DashboardSummary启动的生命周期已经结束，主要是完成一些对象的初始化工作（无数据），需要继续分析Setting的onResume周期，最终调用的是SettingsDrawerActivity的**onResume**()方法。

在SettingsDrawerActivity中注册了安装应用状态变化的广播接收器等,但是这里还进行了一个异步操作:

**newCategoriesUpdateTask().execute();**

在这个AsyncTask中,doInBackground()调用了**CategoryManager**的**reloadAllCategories**()函数,而onPostExecute则调用了接口CategoryListener的唯一方法onCategoriesChanged(),那么作为[界面](https://www.baidu.com/s?wd=%E7%95%8C%E9%9D%A2&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)容器的DashboardSummary肯定重载了这个接口,实现了**onCategoriesChanged**()方法:

DashboardSummary在方法**onCategoriesChanged**()中进行了界面的刷新,

**mCategoryManager.reloadAllCategories(SettingsDrawerActivity.this,getSettingPkg());**

reloadAllCategories函数中调用了tryInitCategories（）,此函数中是获取数据以及对数据的处理，来看下函数**tryInitCategories**:

在这里调用了getCategories()方法；

具体获取办法追踪到***frameworks\base\packages\SettingsLib\src\com\android\settingslib\drawer\TileUtils.java***中。

**TileUtils.java:**

**（1）调用getTilesForAction()**

**（2）新建categoryMap集合**

**（3）遍历tiles，判断集合中是否有元素含有tile.category,如果没有就执行createCategory**

**（4）将拥有相同属性category的tile加入到对象DashboardCategorycategory对象的list<tile>集合中（两个相同名字，一个是Tile里面变量，一个是DashboardCategory对象）**

**（5）将categoryMap的值赋值给List<DashboardCategory>cagtories以便执行排序算法**

**（6）遍历cagtories利用Collections函数和比较器TILE\_COMPARATOR将category.tiles按照priority从大到小排序**

**（7）利用Collections函数和比较器CATEGORY\_COMPARATOR将categories按照priority从大到小排序**

**最后将加载的数据绑定到**[**适配器**](https://www.baidu.com/s?wd=%E9%80%82%E9%85%8D%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)**中。**

首先是**getTilesForAction**()方法,组装Intent对象。

然后调用**getTilesForIntent**()方法

**（1）利用PM查询所有含有getTilesForAction()生成的intent对象的ResolveInfo集合（<activity>标签）**

**（2）获取集合中每一个AcitvityManifest配置的meta标签name包含com.android.settings.category的value值**

**（3）把activityname和package生成Intent对象**

**（4）将2步骤的值赋值给tile.category**

**（5）获取action为com.android.settings.action.SETTINGS的intent-filter的priority属性**

**（6）将解析meta-data标签的bundle数据赋值给tile.metaData**

*注意：会过滤掉非系统级应用的数据！*

*在此方法中会调用updateTileData()方法*

**（1）解析meta-data标签获取name为META\_DATA\_PREFERENCE\_ICON的value值赋值给icon**

**（2）解析meta-data标签获取name为META\_DATA\_PREFERENCE\_TITLE的resource值赋值给title**

**（3）解析meta-data标签获取name为META\_DATA\_PREFERENCE\_SUMMARY的value值赋值给summary**

**（4）如果title为空就获取acitvity标签label属性赋值给title**

**（5）如果icon等于0就获取acitvity标签icon属性赋值给icon**

**createCategory**()方法创建Category。

**（1）创建DashboardCategory对象**

**（2）利用PM查询所有含有Tile对象categoriyKey生成的intent对象的ResolveInfo集合**

**（3）把acitivitylabel值赋值给categorytitle属性**

**（4）把解析intent-filter标签的priority值赋值给category属性**

对于第一级菜单的加载。在**AndroidManifest.xml**中的配置如下列图：（示例）

配置完之后会在**onCategoriesChanged**()中进行界面的加载。

**由此可以知道第一级菜单完全是*动态加载*！**

**AndroidOSettings源码流程分析（数据加载之二级菜单）**

2018年10月22日21:15:09[又双叒叕Fixed了](https://me.csdn.net/qq_34149526)阅读数：226

**AndroidOSettings**

* [静态界面篇](https://blog.csdn.net/qq_34149526/article/details/83186719)
* [界面渲染篇](https://blog.csdn.net/qq_34149526/article/details/83211665)
* [数据加载篇之一级菜单](https://blog.csdn.net/qq_34149526/article/details/83239567)
* [数据加载篇之二级菜单](https://blog.csdn.net/qq_34149526/article/details/83278597)
* [数据加载篇之获取及修改默认设置属性值](https://blog.csdn.net/qq_34149526/article/details/83307695)
* [搜索栏篇](https://blog.csdn.net/qq_34149526/article/details/83341195)

**数据加载之二级菜单**

二级菜单是**动态加载**和**静态xml布局文件**相结合的方式。

以“**系统**”这一项为例进行分析：

首先是清单文件**AndroidManifest.xml**

所对应的Fragment是**SystemDashboardFragment**.java，它继承**DashboardFragment**.java。

而DashboardFragment继承于**PreferenceFragment**

PreferenceFragment是通过xml文件来进行加载的，**它是所有二级界面的基类。**

**先看静态加载部分：**DashboardFragmen——>**displayResourceTiles**()

先调用**getPreferenceScreenResId**（）获取xml布局文件的资源id，该方法在其子类SystemDashboardFragment中实现：

然后调用**addPreferencesFromResource**（）方法加载该XML文件

（相当于设置布局setContentView）

获取**PreferenceScreen**对象screen用于展示所有的子菜单项

遍历**AbstractPreferenceController**对象集合，获取到所有可展示的子菜单项；并将它添加进Screen对象中用于展示。

该集合是在**SystemDashboardFragment**中实现数据加载的工作；

*下图是通过log打印出的“系统”项下所有可展示的子菜单：*

**注：AbstractPreferenceController是用于对所有菜单项进行统一管理的一个抽象类（例如展示或隐藏，监听点击事件等）；**

有几个重要的方法需要被重写：

1、**getPreferenceKey**()用于获取到XML文件中每个Preference节点对应的key值；

2、**isAvailable**（）用于判断该菜单项是否可以显示

3、**updateState**()用于当设置页面选项发生改变时

**再看动态加载部分：**DashboardFragmen——>**refreshDashboardTiles**(）

首先是利用**mDashboardFeatureProvider**对象获取子菜单所属类别（例如：系统）；

**getCategoryKey**()从DashboardFragmentRegistry.PARENT\_TO\_CATEGORY\_KEY\_MAP中获取Category值。

该值通过**类名**获取

存：PARENT\_TO\_CATEGORY\_KEY\_MAP.put(**SystemDashboardFragment.class.getName(),CategoryKey.CATEGORY\_SYSTEM**);

CATEGORY\_SYSTEM="**com.android.settings.category.ia.system**";

然后获取该类别下所有的**子菜单项**；

然后创建一个List用于**保存那些被移除的菜单项**；

接下来就是**遍历获取的tiles集合**，

通过判断key值，是否为空，是否已经被包含（mDashboardTilePrefKeys集合）

对每一个符合要求的tile与Preference通过bindPreferenceToTile()进行绑定；

该方法实现是在：***packages\apps\Settings\src\com\android\settings\dashboard\DashboardFeatureProviderImpl.java***->**bindPreferenceToTile**()

最终将界面显示出来；

**AndroidOSettings源码流程分析（数据加载之获取及修改默认设置属性值）**

2018年10月23日15:36:39[又双叒叕Fixed了](https://me.csdn.net/qq_34149526)阅读数：152

**AndroidOSettings**

* [静态界面篇](https://blog.csdn.net/qq_34149526/article/details/83186719)
* [界面渲染篇](https://blog.csdn.net/qq_34149526/article/details/83211665)
* [数据加载篇之一级菜单](https://blog.csdn.net/qq_34149526/article/details/83239567)
* [数据加载篇之二级菜单](https://blog.csdn.net/qq_34149526/article/details/83278597)
* [数据加载篇之获取及修改默认设置属性值](https://blog.csdn.net/qq_34149526/article/details/83307695)
* [搜索栏篇](https://blog.csdn.net/qq_34149526/article/details/83341195)

**加载默认设置值及修改：（涉及SettingsProvider）**

**示例**：(设备自动亮度设置)——在**Display**菜单下：

其对应的Fragment是：**DisplaySettings.java**；

加载布局：**display\_settings.xml**：

……

……

所有的菜单项都在这里；

而其中一项<SwitchPreference>则对应的是**自动亮度（auto\_brightness）**选项：

接下来看**DisplaySettings**.java——>**getPreferenceControllers**():

获取该项对应的**controller**对象并将它添加到controllers中并返回；

该方法返回**buildPreferenceControllers**():

……

通过AutoBrightnessPreferenceController这一项来分析一下设置值是如何加载以及如何修改；

**AutoBrightnessPreferenceController**.java:

自动亮度值的**获取**：

自动亮度值的**修改**：

获取和修改的实质是对**SettingsProvider**的操作：

（framework有一个类**Settings.java**对使用***SettingsProvider***进行了封装。）

**AndroidOSettings源码流程分析（搜索栏篇（上））**

2018年10月24日11:32:13[又双叒叕Fixed了](https://me.csdn.net/qq_34149526)阅读数：189

**AndroidOSettings**

* [静态界面篇](https://blog.csdn.net/qq_34149526/article/details/83186719)
* [界面渲染篇](https://blog.csdn.net/qq_34149526/article/details/83211665)
* [数据加载篇之一级菜单](https://blog.csdn.net/qq_34149526/article/details/83239567)
* [数据加载篇之二级菜单](https://blog.csdn.net/qq_34149526/article/details/83278597)
* [数据加载篇之获取及修改默认设置属性值](https://blog.csdn.net/qq_34149526/article/details/83307695)
* [搜索栏篇](https://blog.csdn.net/qq_34149526/article/details/83341195)

**Settings搜索栏**

* [上篇——界面](https://blog.csdn.net/qq_34149526/article/details/83341195)
* [中篇——实现原理](https://blog.csdn.net/qq_34149526/article/details/83378184)
* [下篇——搜索及页面跳转](https://blog.csdn.net/qq_34149526/article/details/83414364)

**上篇——界面**

首先看一下搜索框的**布局**实现；

位于Settings主界面布局文件**settings\_main\_dashboard**.xml中：

从布局中并不能看出搜索栏是怎么展现出来的；

接着分析**SettingsActivity**.java——>**onCreate**():

这段代码的含义就是当判断当前界面为Settings主界面时，就会将主界面最上端标题栏替换为**search\_edittext**.xml的界面显示；

这个才是真正的搜索栏。

毫无疑问，会给整个搜索栏**设置监听事件**：

当点击搜索栏时会跳转到**SearchActivity**.java:

看它所加载的布局**search\_main**.xml:

仅仅有一个FrameLayout，不包含任何内容；

来到**SearchFragment**.java:

接下来看它加载的布局**search\_panel**.xml：

……

……

……

此界面所展示的顶部是一个**SearchView**搜索框，而它的下面则是一个**RecyclerView**用于展示搜索候选项；

综上意为：当用户点击主界面的搜索栏时会跳转到一个新的Fragment展示出一个顶部搜索栏，下面候选项列表的新界面。

**关于界面显示已经分析完毕；**

**AndroidOSettings源码流程分析（搜索栏篇（中））**

2018年10月25日13:31:00[又双叒叕Fixed了](https://me.csdn.net/qq_34149526)阅读数：229

**AndroidOSettings**

* [静态界面篇](https://blog.csdn.net/qq_34149526/article/details/83186719)
* [界面渲染篇](https://blog.csdn.net/qq_34149526/article/details/83211665)
* [数据加载篇之一级菜单](https://blog.csdn.net/qq_34149526/article/details/83239567)
* [数据加载篇之二级菜单](https://blog.csdn.net/qq_34149526/article/details/83278597)
* [数据加载篇之获取及修改默认设置属性值](https://blog.csdn.net/qq_34149526/article/details/83307695)
* [搜索栏篇](https://blog.csdn.net/qq_34149526/article/details/83341195)

**Settings搜索栏**

* [上篇——界面](https://blog.csdn.net/qq_34149526/article/details/83341195)
* [中篇——实现原理](https://blog.csdn.net/qq_34149526/article/details/83378184)
* 下[篇——搜索及页面跳转](https://blog.csdn.net/qq_34149526/article/details/83414364)

**中篇——实现原理**

**初始化候选项：**

***相关类：***

***/packages/apps/Settings/src/com/android/settings/SettingsActivity.java***

***/packages/apps/Settings/src/com/android/settings/search/SearchActivity.java***

***/packages/apps/Settings/src/com/android/settings/search/SearchFragment.java***

***/packages/apps/Settings/src/com/android/settings/search/SearchFeatureProviderImpl.java***

***/packages/apps/Settings/src/com/android/settings/search/DatabaseIndexingManager.java***

***/packages/apps/Settings/src/com/android/settings/search/IndexDatabaseHelper.java***

***/packages/apps/Settings/src/com/android/settings/search/SearchIndexableResources.java***

***/frameworks/base/core/java/android/provider/SearchIndexableResource.java***

***/packages/apps/Settings/src/com/android/settings/search/SettingsSearchIndexablesProvider.java***

***/frameworks/base/core/java/android/provider/SearchIndexablesProvider.java***

首先来分析系统设置可以对哪些设置项进行构建搜索数据库，并且是如何加载的；

**SearchFragment**.java——>**onCreate**():

……

**SearchFeatureProviderImpl**.java——>**updateIndexAsync**():

**getIndexingManager**():

**DatabaseIndexingManager**.java——>**indexDatabase**():

可以看到在方法中开启了一个线程任务**IndexingTask**：

它在后台所执行的方法**performIndexing**()：

首先查找系统中所有的配置了"***android.content.action.SEARCH\_INDEXABLES\_PROVIDER***"的Provider

添加其他APP的设置项：**addIndexablesFromRemoteProvider**();

添加其他APP中不需要被搜索到的设置项：**addNonIndexablesFromRemoteProvider**();

上面的addIndexablesFromRemoteProvider会**添加设置项到内存中的一个mDataToProcess对象里**，然后**updateDate**()将该对象更新到数据库中

来看**addIndexablesFromRemoteProvider**()方法：

* 根据当前包名创建对应包的context对象。
* 根据当前包名构建指定URI，例如，
* settings：content://com.android.settings/settings/indexables\_xml\_res
* 然后通过context对象查找对应的Provider的数据
* 所有的要被列入搜索项的所在的APP的Provider都要继承SearchIndexablesProvider
* SearchIndexablesProvider继承ContentProvider；

……

在该类中定义了查询路径并且实现通过匹配不同Uri进行查找；

而在Settings中，则是由**SettingsSearchIndexablesProvider**实现相应查询逻辑

在该类中采用Matrix构建虚拟的数据表，并通过**SearchIndexableResource.values**获取到所有添加到map集合中的SearchIndexableResource;

回到**addIndexablesFromRemoteProvider**();

看一下方法**addIndexablesForXmlResourceUri**():

首先获取到指定包对应的**ContentResolver**，然后解析Cursor数据并且添加到**UpdateData**对象的**dataToUpdate**属性中，该属性是一个List集合；

回到**performIndexing**()继续分析

然后执行方法**updateDatabase**():

接着看方法**addDataToDatabase**():（**加载数据到搜索数据库中**）

看它里面的重要方法**indexOneSearchIndexableData**():

两种添加到数据库中的方法；

接着看方法**indexOneResource**()：

……

……

……

**indexFromResource**()-->使用**XmlResourceParser**解析xml布局

……

……

**indexFromProvider**()-->获取需要解析的布局，再次调用**indexFromResource**()方法；

……

**updateOneRowWithFilteredData**()-->**updateOneRow**()-->此方法最终将解析的数据更新至数据库

在此过程中涉及到了一个组装数据的类**SearchIndexableResources**.java

例如将WiFi相关设置项加入到搜索数据匹配库：

添加有两种方式：

一种是直接在newSearchIndexableResource()时**传入布局文件**；

另一种为"**NO\_DATA\_RES\_ID**"表示此搜索项匹配没有需要解析的xml文件,此xml的解析在**DatabaseIndexingManager**.java中,

采用第二种时,需要在对应的类中创建一个**SEARCH\_INDEX\_DATA\_PROVIDER**,类型为SearchIndexProvider,继承BaseSearchIndexProvider并复写其两个方法:**getXmlResourcesToIndex**()和**getNonIndexableKeys**().

这些“WiFi”候选项被存入一个**HashMap**中：

以候选项类名作为key,把每个候选项封装成SearchIndexableResource对象作为value;

而SearchIndexableResource是继承于SearchIndexableData:

而它是出现在addDataToDatabase()中循环遍历的对象，然后添加至数据库中；

再次回到方法**performIndexing**()，很显然一个和数据库相关的类：IndexDatabaseHelper.java:

……

通过名字可以看出数据库search\_index.db是与搜索索引相关的数据库

数据库路径:***/data/user\_de/0/com.android.settings/databases/search\_index.db***

search\_index.db数据库的**prefs\_index**表格中存放的就是搜索的设置选项

**注：该数据库的初始化不是在开机时，而是在每一次打开Settings或者切换用户时进行。（系统为每一个用户维护一个单独的search\_index）**

至此，Settings搜索框初始化搜索索引完成；

**AndroidOSettings源码流程分析（搜索栏篇（下））**

2018年10月26日14:22:48[又双叒叕Fixed了](https://me.csdn.net/qq_34149526)阅读数：104

**AndroidOSettings**

* [静态界面篇](https://blog.csdn.net/qq_34149526/article/details/83186719)
* [界面渲染篇](https://blog.csdn.net/qq_34149526/article/details/83211665)
* [数据加载篇之一级菜单](https://blog.csdn.net/qq_34149526/article/details/83239567)
* [数据加载篇之二级菜单](https://blog.csdn.net/qq_34149526/article/details/83278597)
* [数据加载篇之获取及修改默认设置属性值](https://blog.csdn.net/qq_34149526/article/details/83307695)
* [搜索栏篇](https://blog.csdn.net/qq_34149526/article/details/83341195)

**Settings搜索栏**

* [上篇——界面](https://blog.csdn.net/qq_34149526/article/details/83341195)
* [中篇——实现原理](https://blog.csdn.net/qq_34149526/article/details/83378184)
* [下篇——搜索及页面跳转](https://blog.csdn.net/qq_34149526/article/details/83414364)

**中篇——搜索及页面跳转**

在搜索过程中，会从数据库中查找匹配，点击筛选结果，根据ClassName启动对应的界面，代码实现回到**SearchFragment**.java——>**onCreate**()中：

……

初始化搜索结果显示适配器SearchResultAdapter；

接下来看**onCreateView**():

将**RecyclerView**与**Adapter**进行绑定；

然后进入到SearchResultAdapter.java:

里面包含一些结果集对象，再看与Adapter相关的ViewHolder对象：

通过判断不同结果类型，加载不同view；

以**IntentSearchViewHolder**为例，看一下它里面的**onBind**（）方法；

很容易的可以看出，它为每一个搜索结果候选项设置**设置点击监听事件**，获取到该项结果中的**intent**对象，调用**startActivity**实现结果页面的跳转。