Contents

[Android Studio 2](#_Toc442449983)

[<Android编程权威指南> 2](#_Toc442449984)

[界面布局 2](#_Toc442449985)

[子页面Fragment动态切换(类似WPF的子控件) 3](#_Toc442449986)

[两个页面叠加互相切换ViewSwitcher 5](#_Toc442449987)

[RESTful请求 6](#_Toc442449988)

[方式一Spring RestTemplate 6](#_Toc442449989)

[方式二Retrofit 7](#_Toc442449990)

[方式三Volley 8](#_Toc442449991)

[Notification and Pending Intent 8](#_Toc442449992)

[Service 9](#_Toc442449993)

[数据库和content provider 10](#_Toc442449994)

[数据库SQLite 10](#_Toc442449995)

[//数据仓库方法一：类似Spring Data Repository 11](#_Toc442449996)

[//数据仓库方法二：content provider等效于数据仓库接口 12](#_Toc442449997)

[<BAMA Android客户端> 16](#_Toc442449998)

[Application 16](#_Toc442449999)

[Context 17](#_Toc442450000)

[SharedPreferences 18](#_Toc442450001)

[SystemService 19](#_Toc442450002)

[消息传递 20](#_Toc442450003)

[网络请求 (Volley) 21](#_Toc442450004)

[解决软件盘问题 21](#_Toc442450005)

[Android同时监听所有Activity生命周期性状态 21](#_Toc442450006)

[BAMA Android客户端 -UML 23](#_Toc442450007)

[<Android开发秘籍> 25](#_Toc442450008)

[复用自组件，改变控件样式 26](#_Toc442450009)

[后台服务 27](#_Toc442450010)

[Android应用启动web:隐式Intent 和使用控件WebView 30](#_Toc442450011)

[自定义组件 (具体参考教材) 31](#_Toc442450012)

[获取设备地理位置信息 32](#_Toc442450013)

[SQLite本地数据库 32](#_Toc442450014)

[异步加载数据 34](#_Toc442450015)

[<eCareCompanion> 36](#_Toc442450016)

# Android Studio

Alt + 6 open logcat window

# <Android编程权威指南>

## 界面布局

LinearLayout：线性布局

可分为垂直布局（android:orientation="vertical"）和水平布局（android:orientation="horizontal" ），在LinearLayout里面可以放多个控件，但是一行（列）只能放一个控件。

RelativeLayout：相对布局

其子控件是根据所设置的参照控件来进行布局的，设置的参照控件可以是父控件，也可以使其他的子控件。

RelativeLayout用到的一些重要的属性：

第一类:属性值为true或false

android:layout\_centerHrizontal 水平居中

android:layout\_centerVertical 垂直居中

android:layout\_centerInparent 相对于父元素完全居中

android:layout\_alignParentBottom 贴紧父元素的下边缘

android:layout\_alignParentLeft 贴紧父元素的左边缘

android:layout\_alignParentRight 贴紧父元素的右边缘

android:layout\_alignParentTop 贴紧父元素的上边缘

android:layout\_alignWithParentIfMissin如果对应的兄弟元素找不到的话就以父元素做参照物

第二类：属性值必须为id的引用名“@id/id-name”

android:layout\_below 在某元素的下方

android:layout\_above 在某元素的的上方

android:layout\_toLeftOf 在某元素的左边

android:layout\_toRightOf 在某元素的右边

android:layout\_alignTop 本元素的上边缘和某元素的的上边缘对齐

android:layout\_alignLeft 本元素的左边缘和某元素的的左边缘对齐

android:layout\_alignBottom 本元素的下边缘和某元素的的下边缘对齐

android:layout\_alignRight 本元素的右边缘和某元素的的右边缘对齐

第三类：属性值为具体的像素值，如30dip，40px

android:layout\_marginBottom 离某元素底边缘的距离

android:layout\_marginLeft 离某元素左边缘的距离

android:layout\_marginRight 离某元素右边缘的距离

android:layout\_marginTop 离某元素上边缘的距离

TableLayout：表格布局

以行列的形式来管理子控件的，在表格布局中的每一行可以是一个View控件或者是一个TableRow控件。而TableRow控件中还可以添加子控件。

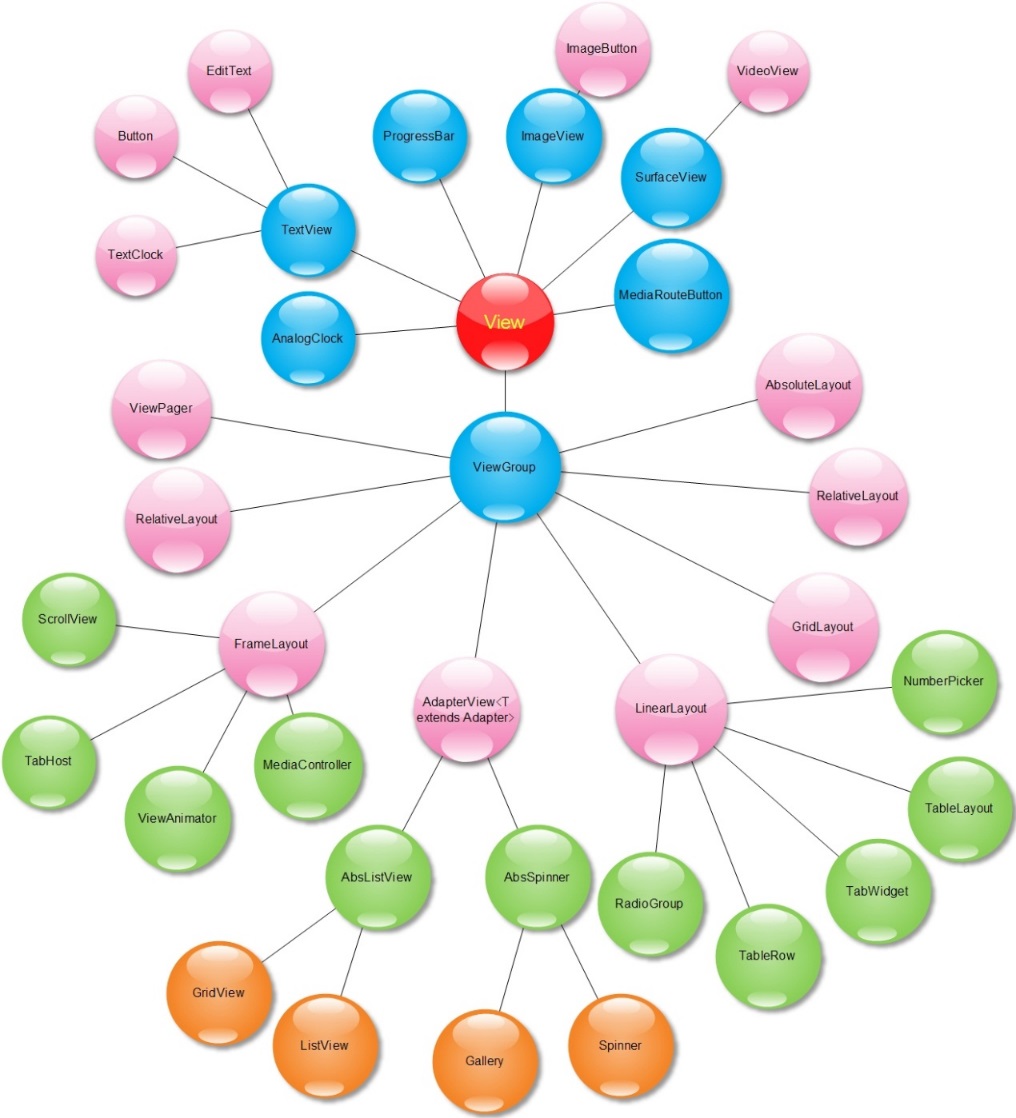
FrameLayout：单帧布局

所有控件都放置在屏幕左上角（0,0），可以放多个控件，但是会按控件定义的先后顺序依次覆盖，后一个会直接覆盖在前一个之上显示，如果后放的比之前的大，会把之前的全部盖住（类似于一层层的纸张）。

AbsoluteLayout：绝对布局

可以直接指定子控件的绝对位置（例如： android:layout\_x="60px" android:layout\_y="32px" ），这种布局简单直接，但是由于手机的分辨率大小不统一，绝对布局的适应性比较差。

布局一览表



## 子页面Fragment动态切换(类似WPF的子控件)

父页面:

<fragment

android:name="com.philips.gs\_fragment.ProductlistFragment"

android:id="@+id/products\_fragment"

android:layout\_width="0dp"

android:layout\_height="match\_parent"

android:layout\_weight="1" />

<FrameLayout

android:id="@+id/dynamic\_fragment"

android:layout\_width="0dp"

android:layout\_height="match\_parent"

android:layout\_weight="2">/>

点击左边列表选项，右边子页面动态切换

左边子页面

public interface OnOptionSelectedListener {

void onOptionSelected(String message) ;

}

OnOptionSelectedListener myListener ;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

// Inflate the layout for this fragment

View view = inflater.inflate(R.layout.fragment\_productlist, container, false);

String[] products = {"Camera", "Laptop", "Watch", "Smartphone", "Television"} ;

ArrayAdapter<String> adapter = new ArrayAdapter<String>(

getActivity().getApplicationContext(), android.R.layout.simple\_list\_item\_1, products) ;

ListView productsList = (ListView) view.findViewById(R.id.products\_list) ;

productsList.setAdapter(adapter);

productsList.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

TextView textView = (TextView) view;

String message = textView.getText().toString();

myListener.onOptionSelected(message);

}});

return view ;

}

//强制要求父页面实现接口

@Override public void onAttach(Activity activity){

super.onAttach(activity);

try {

myListener = (OnOptionSelectedListener) activity ;

} catch (ClassCastException e){

throw new ClassCastException(activity.toString()+" must implement onOptionSelected") ;

}

}

父页面

//响应左页面列表选项事件，切换右子页面

@Override public void onOptionSelected(String message) {

if (message.equals("Camera")){

fragment = CameraFragment.getInstance(message) ;

TAG = CameraFragment.TAG ;

}

else {

fragment = ProductFragment.getInstance(message) ;

TAG = ProductFragment.TAG ;

}

FragmentTransaction fragmentTransaction = getFragmentManager().beginTransaction() ;

fragmentTransaction.replace(R.id.dynamic\_fragment, fragment, TAG) ;

fragmentTransaction.commit();

}

## 两个页面叠加互相切换ViewSwitcher

android.widget.ViewSwitcher是ViewAnimator的子类，用于在两个View之间切换，但每次只能显示一个View。若View的数量超过两个，会抛出异常：java.lang.IllegalStateException，打印 "Can't add more than 2 views to a ViewSwitcher"

activity\_main.xml

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical">

<**ViewSwitcher**

android:id="@+id/viewswitcher"

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1">

<**ImageView**

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:src="@mipmap/ic\_launcher" />

<**ImageView**

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:src="@mipmap/ic\_launcher" />

</ViewSwitcher>

<LinearLayout

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="right"

android:orientation="horizontal" >

<Button

android:id="@+id/prev"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="previous" />

<Button

android:id="@+id/next"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="next" />

</LinearLayout>

</LinearLayout>

MainActivity.java

final ViewSwitcher viewSwitcher = (ViewSwitcher) findViewById(R.id.viewswitcher);

//设置切换动画

viewSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,android.R.anim.slide\_in\_left));

viewSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,android.R.anim.slide\_out\_right));

findViewById(R.id.prev).setOnClickListener(new View.OnClickListener() {

@Override public void onClick(View arg0) {

viewSwitcher.showPrevious(); //显示第一个页面

}

});

findViewById(R.id.next).setOnClickListener(new View.OnClickListener() {

@Override public void onClick(View arg0) {

viewSwitcher.showNext(); //显示第二个页面

}

});

## RESTful请求

### 方式一Spring RestTemplate

build.gradle

dependencies {

compile 'org.springframework.android:spring-android-rest-template:2.0.0.M1'

compile 'com.fasterxml.jackson.core:jackson-databind:2.3.2'

}

private class HttpRequestTask extends AsyncTask<Void, Void, Greeting> {

//后台线程抓取网络数据

@Override protected Greeting doInBackground(Void... params) {

try {

final String url = "http://161.92.141.144:8080/greeting";

return fetchDataByRestTemplate(String url);

} catch (Exception e) {

Log.e("MainActivity", e.getMessage(), e);

}

return null;

}

private Greeting fetchDataByRestTemplate(String url){

RestTemplate restTemplate = new RestTemplate();

restTemplate.getMessageConverters().add(new MappingJackson2HttpMessageConverter());

Greeting greeting = restTemplate.getForObject(url, Greeting.class);

return greeting;

}

//返回结果给主线程

@Override protected void onPostExecute(Greeting greeting) {

TextView greetingIdText = (TextView) findViewById(R.id.id\_value);

TextView greetingContentText = (TextView) findViewById(R.id.content\_value);

greetingIdText.setText(greeting.getId());

greetingContentText.setText(greeting.getContent());

}

主线程启动后台

new HttpRequestTask().execute();

### 方式二Retrofit

build.gradle

dependencies {

compile 'com.squareup.retrofit:retrofit:1.9.0'

}

//API接口GreetingService.java

public interface GreetingService {

@GET("/") Greeting greeting(@Query("name")String name) ;

@GET("/") void greeting(@Query("name")String name, Callback<Greeting> callback) ;

}

//Retrofit封装后台线程，去访问RESTful API

RestAdapter restAdapter = new RestAdapter.Builder().setEndpoint(url).build() ;

GreetingService service = restAdapter.create(GreetingService.class) ;

service.greeting("QiZhong Lin", new Callback<Greeting>() {

//异步返回结果给主线程

@Override public void success(Greeting greeting, Response response) {

updateActivity(greeting);

}

@Override public void failure(RetrofitError error) { }

});

### 方式三Volley

build.gradle

dependencies {

compile 'com.mcxiaoke.volley:library:1.0.+'

}

RequestQueue requestQueue = Volley.newRequestQueue(context) ;

JsonObjectRequest request = new JsonObjectRequest(url, null, new Response.Listener<JSONObject>() {

@Override public void onResponse(JSONObject response) {

Greeting greeting = new Gson().fromJson(response.toString(), Greeting.class) ;

updateActivity(greeting);

}}, new Response.ErrorListener(){

@Override public void onErrorResponse(VolleyError error) {

Log.e("TAG", error.getMessage(), error);

}

}) ;

requestQueue.add(request) ;

## Notification and Pending Intent

Notification显示为通知区域内的一个图标，用户打开通知来查看通知的内容，点击某条通知，将会调用预设的Pending Intent Activity来执行所需的Action.

Pending Intent:正如其名，是一个被保存起来并随时准备在将来执行的Intent

Android的通知系统：用户可以通过文字，震动，指示灯闪动，声音提醒来获得通知，通知永远不会自动启动一个Activity,它只是向用户报告一下，只有选中某通知时才会启动Activity.

//创建Pending Intent

Intent intent = new Intent(MainActivity.this, WelcomeActivity.class) ;

PendingIntent pendingIntent = PendingIntent.getActivity(MainActivity.this, 0, intent, 0) ;

//创建通知

Notification.Builder builder = new Notification.Builder(MainActivity.this)

.setSmallIcon(R.drawable.addpatient)

.setAutoCancel(true)

.setTicker("Notification to launch Pending Intent")

.setWhen(System.currentTimeMillis())

.setContentTitle("Message")

.setContentText("Let us launch the pending Intent")

.setContentIntent(pendingIntent) ;

Notification notification = builder.build() ;

//发布通知(系统会显示并管理通知)

NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION\_SERVICE) ;

notificationManager.notify(0, notification);

//此时系统栏中显示出一段文字表示通知已发生，当展开通知后，包括图标、标题和文字内容的详细信息将会显示出来。轻点通知后，Pending Intent将会启动一个预设的Activity

## Service

Service（服务）是一个没有用户界面的在后台运行执行耗时操作的应用组件。其他应用组件能够启动Service，并且当用户切换到另外的应用场景，Service将持续在后台运行。另外，一个组件能够绑定到一个service与之交互（IPC机制），例如，一个service可能会处理网络操作，播放音乐，操作文件I/O或者与内容提供者（content provider）交互，所有这些活动都是在后台进行。

Service有两种状态，“启动的”和“绑定”

启动:startService()

通过startService()启动的服务处于“启动的”状态，一旦启动，service就在后台运行，即使启动它的应用组件已经被销毁了。通常started状态的service执行单任务并且不返回任何结果给启动者。比如当下载或上传一个文件，当这项操作完成时，service应该停止它本身。

绑定：bindService()

还有一种“绑定”状态的service，通过调用bindService()来启动，一个绑定的service提供一个允许组件与service交互的接口，可以发送请求、获取返回结果，还可以通过跨进程通信来交互（IPC）。绑定的service只有当应用组件绑定后才能运行，多个组件可以绑定一个service，当调用unbind()方法时，这个service就会被销毁了。

service与activity一样都存在与当前进程的主线程中，所以，一些阻塞UI的操作，比如耗时操作不能放在service里进行，比如另外开启一个线程来处理诸如网络请求的耗时操作。如果在service里进行一些耗CPU和耗时操作，可能会引发ANR警告，这时应用会弹出是强制关闭还是等待的对话框。所以，对service的理解就是和activity平级的，只不过是看不见的，在后台运行的一个组件，这也是为什么和activity同被说为Android的基本组件。

## 数据库和content provider

### 数据库SQLite

轻量级(仅一个动态库且尺寸小)，独立性（SQLite 数据库的核心引擎不需要依赖第三方软件），隔离性(SQLite 数据库中所有的信息（比如表、视图、触发器等）都包含在一个文件夹内)，跨平台(SQLite 目前支持大部分操作系统，电脑os or Android os)；多语言接口(SQLite 数据库支持多语言编程接口); 安全性(SQLite 数据库通过数据库级上的独占性和共享锁来实现独立事务处理。这意味着多个进程可以在同一时间从同一数据库读取数据，但只能有一个可以写入数据)。

//创建，更新数据库

public class MySqliteOpenHelper extends SQLiteOpenHelper {

//数据库名

private static final String DATABASE\_NAME = "comments.db" ;

private static final int DATABASE\_VERSION = 1 ;

//数据库表及字段

public static final String TABLE\_COMMENTS = "comments" ;

public static final String COLUMN\_ID = "\_id";

public static final String COLUMN\_COMMENT = "comment";

// 创建数据库表schema

private static final String DATABASE\_CREATE = "create table "

+ TABLE\_COMMENTS + "(" + COLUMN\_ID

+ " integer primary key autoincrement, " + COLUMN\_COMMENT

+ " text not null);";

public MySqliteOpenHelper(Context context){

super(context, DATABASE\_NAME, null, DATABASE\_VERSION) ;

}

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL(DATABASE\_CREATE);

}

@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

db.execSQL("DROP TABLE IF EXISTS " + TABLE\_COMMENTS);

onCreate(db);

}

}

### //数据仓库方法一：类似Spring Data Repository

public class CommentsDataSource {

private MySqliteOpenHelper dbHelper ;

private SQLiteDatabase database ;

private String[] allColumns = {MySqliteOpenHelper.COLUMN\_ID, MySqliteOpenHelper.COLUMN\_COMMENT} ;

public CommentsDataSource(Context context){

dbHelper = new MySqliteOpenHelper(context) ;

}

public void open() throws SQLException {

database = dbHelper.getWritableDatabase() ;

}

public void close() {

dbHelper.close();

}

public List<Comment> getAllComments(){

List<Comment> comments = new ArrayList<Comment>() ;

Cursor cursor = database.query(MySqliteOpenHelper.TABLE\_COMMENTS, allColumns, null, null, null, null, null) ;

for (cursor.moveToFirst(); !cursor.isAfterLast(); cursor.moveToNext() ){

comments.add(cursorToComment(cursor)) ;

}

cursor.close();

return comments ;

}

public Comment create(String comment){

ContentValues values = new ContentValues() ;

values.put(MySqliteOpenHelper.COLUMN\_COMMENT, comment);

long insertId = database.insert(MySqliteOpenHelper.TABLE\_COMMENTS, null, values) ;

String[] projection = allColumns ;

String criteria = MySqliteOpenHelper.COLUMN\_ID + "=" + insertId ;

String[] criteriaValues = null ;

String groupBy = null ;

String having = null ;

String orderBy = null ;

Cursor cursor = database.query(MySqliteOpenHelper.TABLE\_COMMENTS, projection, criteria, criteriaValues, groupBy, having, orderBy) ;

cursor.moveToFirst() ;

Comment newComment = this.cursorToComment(cursor) ;

cursor.close();

return newComment ;

}

public void delete(Comment comment){

long id = comment.getId() ;

String criteria = MySqliteOpenHelper.COLUMN\_ID + "=" + id ;

String[] criteriaValues = null ;

database.delete(MySqliteOpenHelper.TABLE\_COMMENTS, criteria, criteriaValues) ;

}

private Comment cursorToComment(Cursor cursor) {

Comment comment = new Comment();

comment.setId(cursor.getLong(0));

comment.setComment(cursor.getString(1));

return comment;

}

}

//客户端使用如在Activity中

private CommentsDataSource dataSource ;

@Override protected void onCreate(Bundle savedInstanceState) {

...

dataSource = new CommentsDataSource(this) ;

dataSource.open();

List<Comment> values = dataSource.getAllComments() ;

...

}

@Override protected void onResume() {

dataSource.open();

super.onResume();

}

@Override protected void onPause() {

dataSource.close();

super.onPause();

}

### //数据仓库方法二：content provider等效于数据仓库接口

**ContentProvider：为存储和获取数据提供统一的接口**。可以在不同的应用程序之间共享数据。Android 系统为一些常见的数据类型(如音乐、视频、图像、手机通信录联系人信息等)内置了一系列的 Content Provider, 这些都位于android.provider包下。持有特定的许可，可以在自己开发的应用程序中访问这些Content Provider。

对于Content Provider，最重要的就是数据模型(data model) 和 URI. Data model: Content Provider 存储的数据以数据表的形式,每一条数据记录都包括一个 "\_ID" 数值字段，改字段唯一标识一条数据; URI:每一个Content Provider 都对外提供一个能够唯一标识自己数据集(data set)的公开URI, 如果一个Content Provider管理多个数据集，其将会为每个数据集分配一个独立的URI

具体例子参考项目BAMA的Android客户端

//content provider 实现getType, onCreate, query, insert, update, delete等操作

public class CommentContentProvider extends ContentProvider {

private static final String Authority = "com.philips.gs\_sqlite\_contentprovider\_loader" ;

//uri <= 数据库表路径 data/data/com…/table

public static final Uri CONTENT\_URI = Uri.parse("content://" + Authority + "/" + MySqliteOpenHelper.TABLE\_COMMENTS) ;

public static final String CONTENT\_TYPE = ContentResolver.CURSOR\_DIR\_BASE\_TYPE + MySqliteOpenHelper.TABLE\_COMMENTS ;

public static final String CONTENT\_TYPE\_ITEM = ContentResolver.CURSOR\_ITEM\_BASE\_TYPE + MySqliteOpenHelper.TABLE\_COMMENTS ;

//uri matcher

private static final UriMatcher uriMatcher = new UriMatcher(UriMatcher.NO\_MATCH) ;

private static final int ALLROWS = 1 ;

private static final int SINGLEROW = 2 ;

static {

uriMatcher.addURI(Authority, MySqliteOpenHelper.TABLE\_COMMENTS, ALLROWS);

uriMatcher.addURI(Authority, MySqliteOpenHelper.TABLE\_COMMENTS+"/#", SINGLEROW);

}

private MySqliteOpenHelper dbHelper ;

@Override

public String getType(Uri uri) {

switch (uriMatcher.match(uri)) {

case ALLROWS:

return CONTENT\_TYPE ;

case SINGLEROW:

return CONTENT\_TYPE\_ITEM ;

default:

throw new IllegalArgumentException("Unsupported URI: " + uri) ;

}

}

@Override

public boolean onCreate() {

dbHelper = new MySqliteOpenHelper(getContext()) ;

return true;

}

@Override

public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {

Cursor cursor = null ;

SQLiteDatabase database = dbHelper.getReadableDatabase() ;

switch (uriMatcher.match(uri)){

case ALLROWS:

cursor = database.query(MySqliteOpenHelper.TABLE\_COMMENTS, null, null, null, null, null, null ) ;

break;

case SINGLEROW:

long \_id = ContentUris.parseId(uri) ;

String criteria = MySqliteOpenHelper.COLUMN\_ID + "=" + \_id ;

if (selection!=null && !selection.equals("")){

criteria += selection ;

}

cursor = database.query(MySqliteOpenHelper.TABLE\_COMMENTS, projection, criteria, selectionArgs, null, null, sortOrder) ;

break ;

default:

break ;

}

cursor.setNotificationUri(getContext().getContentResolver(), uri);

return cursor;

}

@Override

public Uri insert(Uri uri, ContentValues values) {

SQLiteDatabase database = dbHelper.getWritableDatabase() ;

long id = database.insert(MySqliteOpenHelper.TABLE\_COMMENTS, null, values) ;

Uri rUri = ContentUris.withAppendedId(uri, id);

getContext().getContentResolver().notifyChange(rUri, null);

return rUri;

}

@Override

public int delete(Uri uri, String selection, String[] selectionArgs) {

SQLiteDatabase database = dbHelper.getWritableDatabase() ;

int count = 0 ;

switch (uriMatcher.match(uri)){

case ALLROWS:

count = database.delete(MySqliteOpenHelper.TABLE\_COMMENTS, selection, selectionArgs) ;

break;

case SINGLEROW:

long id = ContentUris.parseId(uri) ;

String criteria = MySqliteOpenHelper.COLUMN\_ID + " = " + id + (!TextUtils.isEmpty(selection) ? " AND ( "+selection+")" : "") ;

count = database.delete(MySqliteOpenHelper.TABLE\_COMMENTS, criteria, selectionArgs ) ;

break;

default:

throw new IllegalArgumentException("URI not found: " + uri) ;

}

getContext().getContentResolver().notifyChange(uri, null);

return count ;

}

@Override

public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {

return 0;

}

}

//客户端使用如在Activity中

LoaderManager.LoaderCallbacks<Cursor> loadCallback = new LoaderManager.LoaderCallbacks<Cursor>() {

@Override public Loader<Cursor> onCreateLoader(int id, Bundle args) {

CursorLoader cursorLoader = new CursorLoader(MainActivity.this, CommentContentProvider.CONTENT\_URI, MySqliteOpenHelper.ALL\_COLUMNS, null, null, null) ;

return cursorLoader;

}

@Override public void onLoadFinished(Loader<Cursor> loader, Cursor data) {

ArrayAdapter<Comment> adapter = (ArrayAdapter<Comment>) getListAdapter() ;

adapter.clear();

while (data.moveToNext()){

Comment comment = new Comment();

comment.setId(data.getLong(0));

comment.setComment(data.getString(1));

adapter.add(comment) ;

}

adapter.notifyDataSetChanged();

}

@Override public void onLoaderReset(Loader<Cursor> loader) {

}

} ;

@Override protected void onCreate(Bundle savedInstanceState) {

...

getLoaderManager().initLoader(0, null, loadCallback);...

}

//在其它的响应事件中

//插入数据行

ContentValues values = new ContentValues() ;

values.put(MySqliteOpenHelper.COLUMN\_COMMENT, commentStr) ;

getContentResolver().insert(CommentContentProvider.CONTENT\_URI, values) ;

getLoaderManager().restartLoader(0, null, loadCallback) ;

//删除数据行

Uri uri = Uri.parse(CommentContentProvider.CONTENT\_URI + "/" + comment.getId()) ;

getContentResolver().delete(uri, null, null) ;

//content provider需在AndroidManifest.xml里注册

<provider

android:authorities="com.philips.gs\_sqlite\_contentprovider\_loader"

android:name=".CommentContentProvider"

android:exported="true">

</provider>

# <BAMA Android客户端>

## Application

全局变量, 启动Application时，系统会创建一个PID，即进程ID，所有的Activity都会在此进程上运行。那么我们在Application创建的时候初始化全局变量，同一个应用的所有Activity都可以取到这些全局变量的值

实现：

a)自定义Application

**public** **class** BaseApplication **extends** Application {

**private** **static** **final** String ***LOG\_TAG*** = BaseApplication.**class**.getName();

**private** **static** BaseApplication *appInstance*;

**private** SessionControl sessionControl;

**public** **static** **synchronized** BaseApplication getInstance() {

Log.*d*(***LOG\_TAG***, "Enter into BaseApplication.getInstance.");

**return** *appInstance*;

}

@Override

**public** **void** onCreate() {

Log.*i*(***LOG\_TAG***, "Enter into BaseApplication.onCreate.");

**super**.onCreate();

setSessionControl(**new** SessionControl());

}

}

b)在ApplicationManifest.xml文件中配置自定义的Application

<application

android:name=*"com.philips.pscs.ccs.application.BaseApplication"*

android:allowBackup=*"true"*

android:icon=*"@drawable/icon"*

android:label=*"@string/app\_name"*

android:largeHeap=*"true"*

android:theme=*"@style/AppTheme"* >

<activity

android:name=*"com.philips.pscs.ccs.activity.LoginActivity"*

android:label=*"@string/app\_name"*

android:screenOrientation=*"landscape"*

android:theme=*"@android:style/Theme.Holo.Light.NoActionBar"* >

<intent-filter>

<action android:name=*"android.intent.action.MAIN"* />

<category android:name=*"android.intent.category.LAUNCHER"* />

</intent-filter>

</activity>

</application>

c) 访问Application

//在Activity里

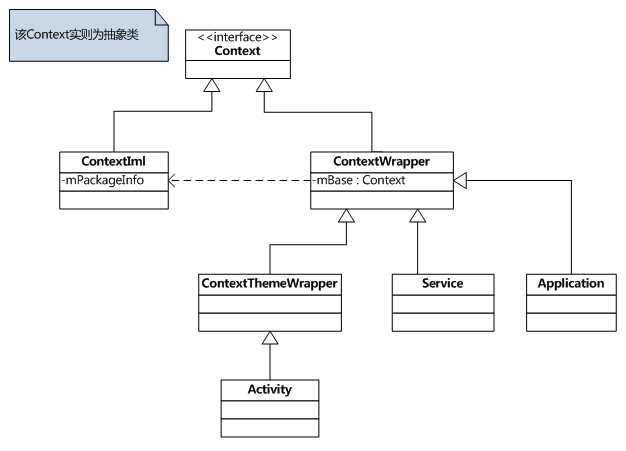
BaseApplication app = (BaseApplication)getApplication();

//任何函数

BaseApplication app = BaseApplication.getInstance() ;

## Context

加载资源、启动一个新的Activity、获取系统服务、获取内部文件路径以及创建view(其实还远不止这些)统统都需要Context对象来完成



各Context能力

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Application | Activity | Service | ContentProvider | BroadcastReceiver |
| Show dialog | N | Y | N | N | N |
| Start activity | N | Y | N | N | N |
| Layout inflation | N | Y | N | N | N |
| Start service | Y | Y | Y | Y | Y |
| Bind to service | Y | Y | Y | Y | N |
| Send broadcast | Y | Y | Y | Y | Y |
| Register broadcast Receiver | Y | Y | Y | Y | N |
| Load resource values | Y | Y | Y | Y | Y |

访问context

getApplicationContext() 返回应用的上下文，生命周期是整个应用，应用摧毁它才摧毁

Activity.this的context 返回当前activity的上下文，属于activity ，activity 摧毁他就摧毁

用法：

基于上下文（一般是父组件）和layout.xml文件实例化组件

View view = LayoutInflater.from(context).inflate(R.layout.tab\_layout,null);

## SharedPreferences

为了保存软件的设置参数，Android 平台为我们提供了一个SharedPreferences 类，它是一个轻量级的存储类，特别适合用于保存软件配置参数。使用SharedPreferences 保存数据，其背后是用xml文件存放数据，文件存放在/data/data/<package name>/shared\_prefs 目录下。

存储数据信息

1）打开名为configuration的配置文件，如果存在则打开它，否则创建新的名为configuration的配置文件

SharedPreferences sharedPreferences = getSharedPreferences("configuration", 0);

2）让sharedPreferences处于编辑状态

SharedPreferences.Editor editor = sharedPreferences.edit();

3）存放数据

editor.putString("name"，"harvey");

4）完成提交

editor.commit();

读取数据信息

1）打开名为configuration的配置文件

SharedPreferences sharedPreferences = getSharedPreferences("configuration", Context.MODE\_PRIVATE);

2）获取数据

String name = sharedPreferences.getString("name","默认值");

以上就是Android中SharedPreferences的使用方法，其中创建的配置文件存放位置可以在Eclipse中查看：

DDMS --- File Explorer --- data/data/<package name>/shared\_prefs/configuration.xml

## SystemService

android的后台运行很多service，它们在系统启动时被SystemServer开启，支持系统的正常工作，比如MountService监听是否有SD卡安装及移除，ClipboardService提供剪切板功能，PackageManagerService提供软件包的安装移除及查看等等



boolean isNetworkConnected = false;

for (NetworkInfo ni : context.getSystemService(Context.CONNECTIVITY\_SERVICE).getAllNetworkInfo()) {

if (ni.isConnected()) isNetworkConnected = true ;

}

获取资源方式

String loginServiceURI = context.getResources().getString(R.string.authloginurl)

## 消息传递

正如其他GUI应用程序一样，Android应用程序也是消息(事件)驱动的。这种消息的传递必须依赖于应用框架提供的消息机制。Android本身提供了2种消息机制：

**· 组件间消息传递 --- Intent**

**· 线程间消息传递 --- Message(集中处理，我更倾向于用AyncTask)**

主线程中所做的操作最好耗时非常短，一般建议创建子线程去完成比较费时的工作(访问网络，下载数据，查询数据库等), 以免主线程阻塞而发生ANR异常。但子线程通常不能更新UI。所以子线程与主线程需要通过Message来沟通信息。这个场景很常见。

主线程创建Handler实例 （主线程）

private Handler mHandler = new Handler(new Handler.Callback() {

public boolean handleMessage(Message msg) {

switch (msg.what) {

case UPDATE\_UI:

{

//update UI

break;

}

default:

break;

}});

子线程利用主线程的Handler向主线程发送消息：(子线程) （这里例子是Volley封装的子线程）

RequestQueue.add(new JsonObjectRequest(Method.POST, url, jsonObj, 响应对象，错误响应对象) ;

new Response.Listener<JSONObject>() {

@Override public void onResponse(JSONObject response) {

BaseReponse baseResponse = JSON.parseObject(reponse.toString(), BaseResponse.class) ;

Message msg = new Message() ;

msg.what = 1;

msg.setData(new Bundle().putSerializable(“baseResponse”, baseResponse) ;

mHandler.sendMessage(msg) ;

}

}

## 网络请求 (Volley)

使用HTTP协议来发送和接收网络数据。Android系统中主要提供了两种方式来进行HTTP通信，HttpURLConnection和HttpClient; 一些Android网络通信框架AsyncHttpClient, Universal-Image-Loader; 在2013年Google I/O大会上推出了一个新的网络通信框架——Volley。Volley可是说是把AsyncHttpClient和Universal-Image-Loader的优点集于了一身. 设计目标就是非常适合去进行数据量不大，但通信频繁的网络操作，而对于大数据量的网络操作，比如说下载文件等，Volley的表现就会非常糟糕

//请求队列

RequestQueue requestQueue = Volley.newRequestQueue(context) ;

//异步的JSON请求

JsonObjectRequest req = new JsonObjectRequest(Method.POST, url, jsonObj, new Response.Listener(){}, new Response.ErrorListener(){}) ;

req.setTag(‘’requestTag”) ; //给该请求赋标签

req.setRetryPolicy(…) ; //设置请求超时，超时后请求次数等

…//还可以设置请求cookies, 头信息等,请查Volley资料

requestQueue.add(req) ;

//取消请求

requestQueue.cancelAll(“requestTag”) ;

## 解决软件盘问题

在做登录和注册页面的时候,经常会遇到诸如软键盘挡住输入框的情况.在你的activity中的oncreate中setContentView之前写上这个代码 (也可在dialog.show之前)

getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT\_INPUT\_STATE\_HIDDEN |WindowManager.LayoutParams.SOFT\_INPUT\_ADJUST\_PAN);

## Android同时监听所有Activity生命周期性状态

在做应用统计时我们可能需要在获取activity的生命周期状态，以前的方式是我们需要重写每一个activity的onResume() 等函数然后埋点，Android 4.0（API Level 14）新增了一个接口ActivityLifecycleCallbacks，用于监听应用中所有Activity的运行情况,解放了开发者统计分析Activity的困难. registerActivityLifecycleCallbacks是Application的一个接口，注册之后应用里的所有activity的生命周期都会被监控起来，我们可以在此接口方法里实现一些特殊的需求。比如统计用户对每个Activity使用情况，我们可以定义一个BaseActivity，在onStart()和onStop()人工插入统计方法。比如从桌面进应用的时候，检测某些状态，就可以在onActivityResumed方法里检测。

//在AndroidManifest里起用自定义Application

<application android:name=".global.BaseApplication"

public class BaseApplication extends Application {

public void onCreate() {

super.onCreate();

this.registerActivityLifecycleCallbacks(new ActivityLifecycleCallbacks() {

@Override public void onActivityStopped(Activity activity) {

Logger.v(activity, "onActivityStopped");

}

@Override public void onActivityStarted(Activity activity) {

Logger.v(activity, "onActivityStarted");

}

@Override public void onActivitySaveInstanceState(Activity activity, Bundle outState) {

Logger.v(activity, "onActivitySaveInstanceState");

}

@Override public void onActivityResumed(Activity activity) {

Logger.v(activity, "onActivityResumed");

}

@Override public void onActivityPaused(Activity activity) {

Logger.v(activity, "onActivityPaused");

}

@Override public void onActivityDestroyed(Activity activity) {

Logger.v(activity, "onActivityDestroyed");

}

@Override public void onActivityCreated(Activity activity, Bundle savedInstanceState) {

Logger.v(activity, "onActivityCreated");

}

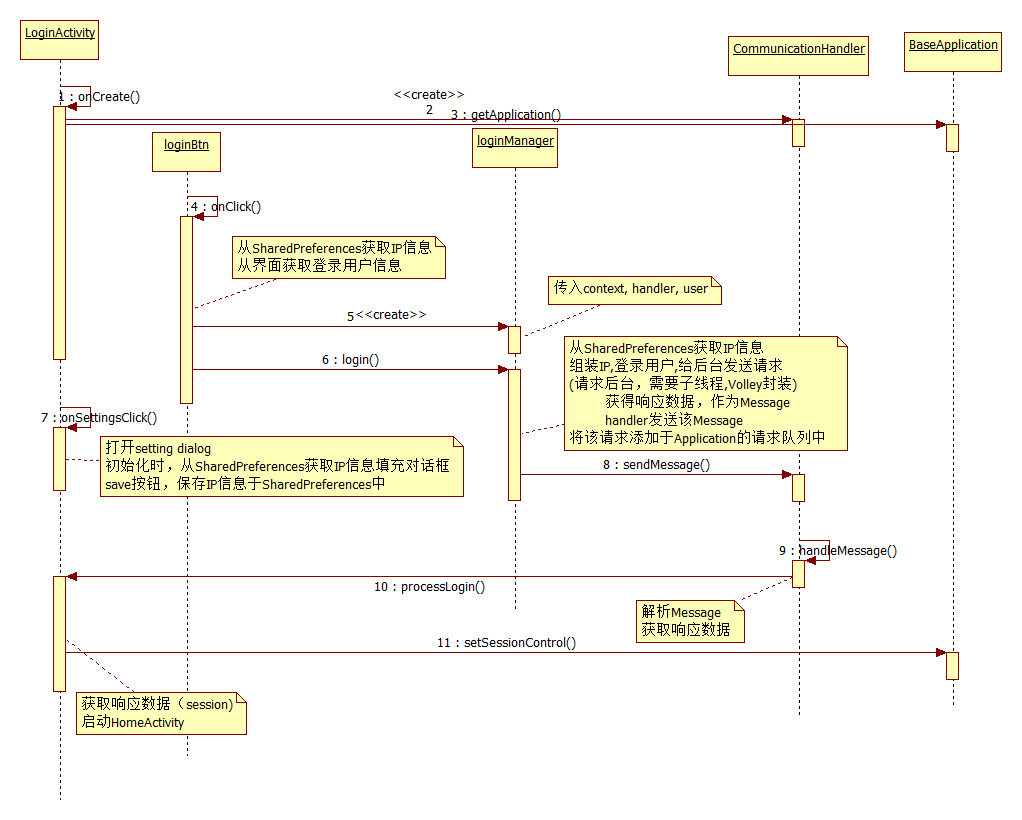
});

};

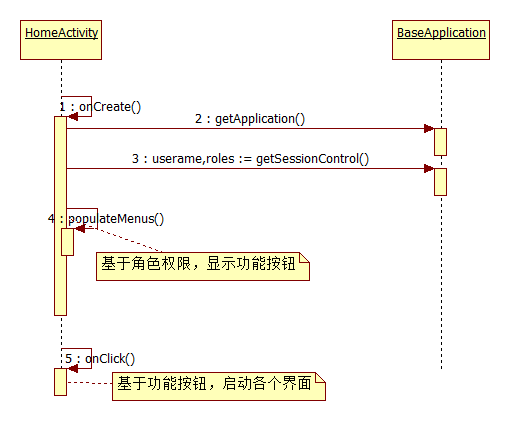
}

## BAMA Android客户端 -UML

LoginActivity.java

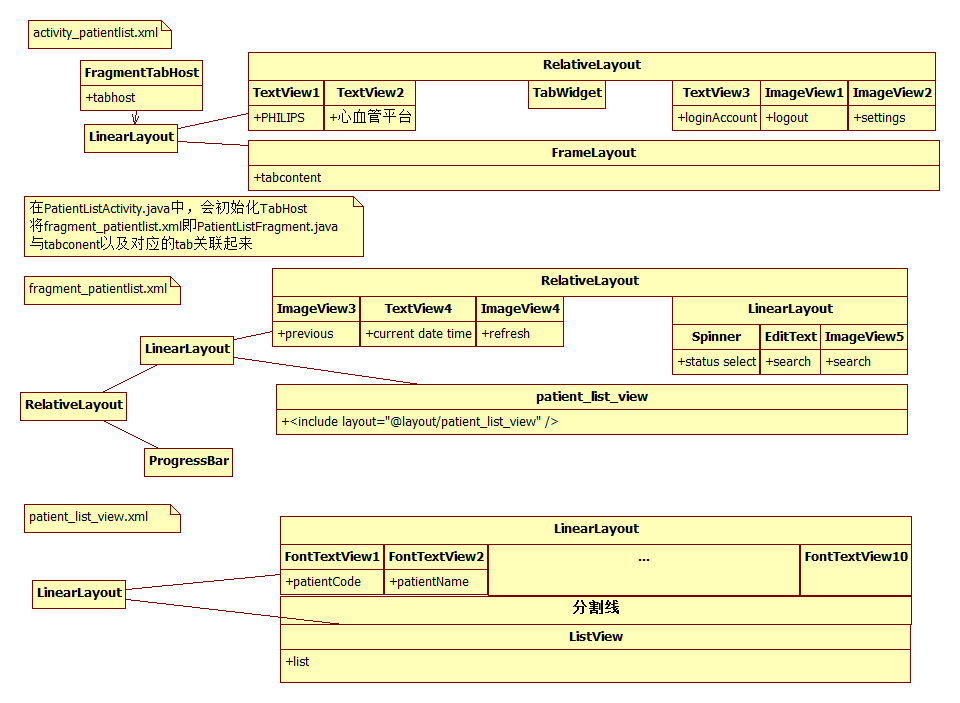


HomeActivity.java

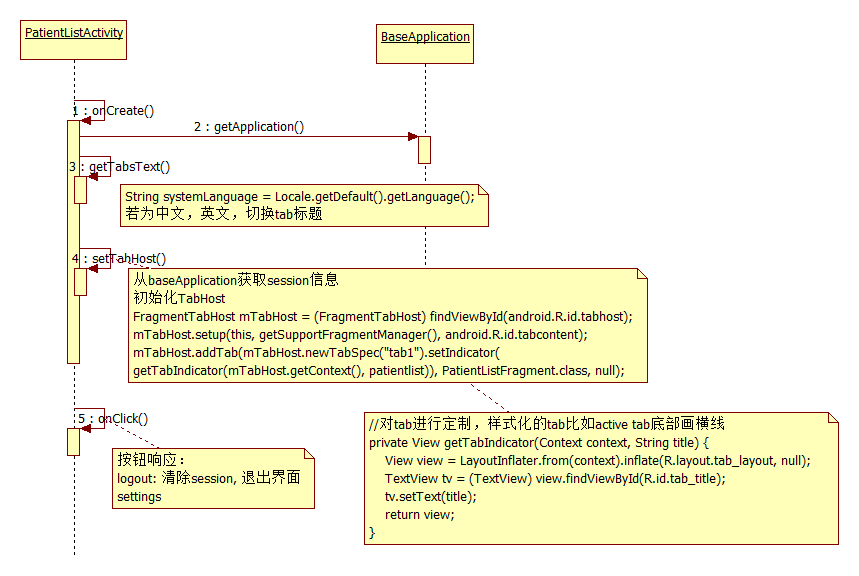


PatientListActivity

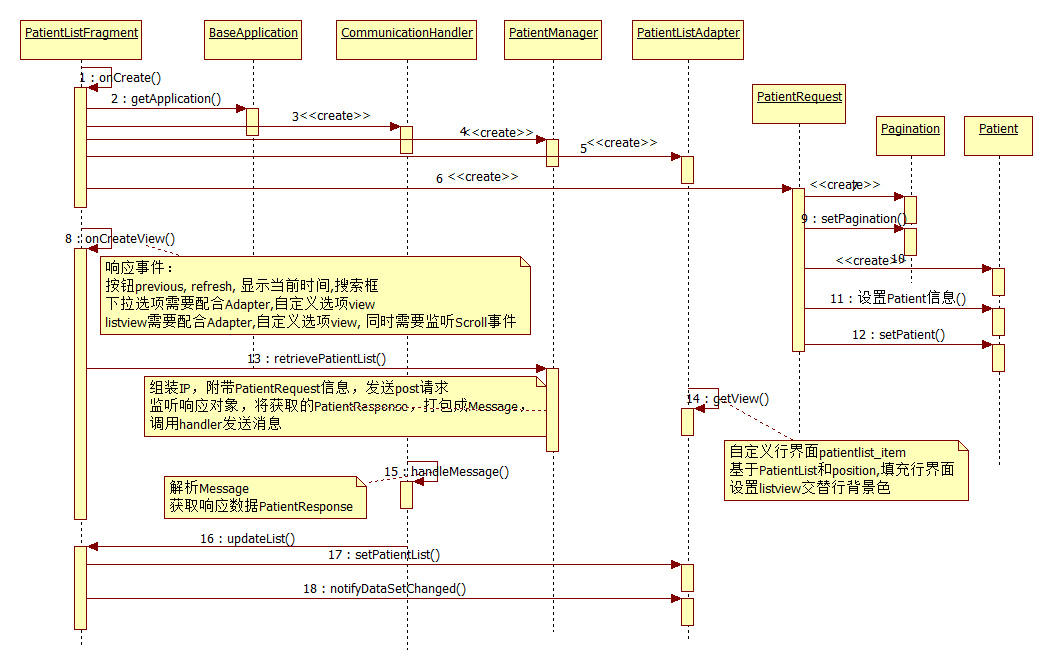
activity\_patientlist.xml + fragment\_patientlist.xml + patient\_list\_view.xml



PatientListActivity.java



PatientListFragment.java



# <Android开发秘籍>

//获取可启动的应用程序主Activity

Intent startupIntent = new Intent(Intent.ACTION\_MAIN);

startupIntent.addCategory(Intent.CATEGORY\_LAUNCHER);

final PackageManager pm = getActivity().getPackageManager();

List<ResolveInfo> activities = pm.queryIntentActivities(startupIntent, 0);

//启动别的应用程序主Activity

Intent i = new Intent(Intent.ACTION\_MAIN);

i.setClassName(activityInfo.applicationInfo.packageName, activityInfo.name);

i.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

startActivity(i);

## 复用自组件，改变控件样式

//复用自定义组件layout/button\_row.xml

<TableLayout android:background="@drawable/remote\_background"

android:stretchColumns="\*" >

<include android:layout\_weight="1" layout="@layout/button\_row" />

<include android:layout\_weight="1" layout="@layout/button\_row" />

</TableLayout>

//自定义组件，引用自定义样式values/styles.xml

<TableRow xmlns:android="http://schemas.android.com/apk/res/android" >

<Button style="@style/RemoteButton" />

<Button style="@style/RemoteButton" />

<Button style="@style/RemoteButton" />

</TableRow>

//自定义样式, 改变背景色和文本颜色（采用drawable/button\_shape\_shadowed.xml…）

<resources xmlns:android="http://schemas.android.com/apk/res/android">

<style name="AppTheme" parent="android:Theme.Light" />

<style name="RemoteButton">

<item name="android:layout\_width">0dp</item>

<item name="android:layout\_height">match\_parent</item>

<item name="android:textSize">20dp</item>

<item name="android:layout\_margin">3dp</item>

<item name="android:background">@drawable/button\_shape\_shadowed</item>

<item name="android:textColor">@drawable/button\_text\_color</item>

</style>

</resources>

//采用layer list drawable and shape drawable 生成阴影色；引用背景色drawable/button\_shape.xml

<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >

<item>

<shape android:shape="rectangle" >

<corners android:radius="5dp" />

<gradient

android:angle="90" android:centerColor="#303339" android:centerY="0.05" android:endColor="#000000" android:startColor="#00000000" />

</shape>

</item>

<item>

<inset android:drawable="@drawable/button\_shape" android:insetBottom="5dp" />

</item>

</layer-list>

// state list drawable: 有状态背景色，按下按钮与松开按钮采用不同的drawable

<selector xmlns:android="http://schemas.android.com/apk/res/android">

<item android:drawable="@drawable/button\_shape\_normal" android:state\_pressed="false"/>

<item android:drawable="@drawable/button\_shape\_pressed"android:state\_pressed="true"/>

</selector>

//按下时的背景色

<shape xmlns:android="http://schemas.android.com/apk/res/android" android:shape="rectangle" >

<corners android:radius="3dp" />

<gradient android:angle="270" android:endColor="#cccccc" android:startColor="#acacac" />

</shape>

//松开时的背景色

<shape xmlns:android="http://schemas.android.com/apk/res/android" android:shape="rectangle" >

<corners android:radius="3dp" />

<gradient android:angle="90" android:endColor="#cccccc" android:startColor="#acacac" />

</shape>

## 后台服务

Activity是Android应用的前台，专注于提供良好的用户视觉体验。服务是Android应用的后台，即使前台关闭，activity长时间停止运行，后台服务依然可以持续不断地执行工作任务。

为保证服务在后台的切实可用，当没有activity在运行时，需通过某种方式在后台执行一些任务。比如设置一个5分钟间隔的定时器。可以采用操作系统内置的系统服务AlarmManager定时发送IntentService服务。

//前端fragment或其他控制层代码启停定时器

Context context = getActivity() ;

Intent i = new Intent(context, PollService.class);

PendingIntent pi = PendingIntent.getService(context, 0, i, 0);

AlarmManager alarmManager = (AlarmManager) context.getSystemService(Context.ALARM\_SERVICE); alarmManager.setRepeating(AlarmManager.RTC, System.currentTimeMillis(), POLL\_INTERVAL, pi);

//IntentService逐个执行命令队列里的命令，并同是为每一条命令在后台线程上调用onHandleIntent(Intent)方法。新进命令总是放置在队列尾部。最后，执行完队列中全部命令后，服务也随即停止并被销毁

public class PollService extends IntentService {

@Override public void onHandleIntent(Intent intent) {

Resources r = getResources();

PendingIntent pi = PendingIntent.getActivity(this, 0, new Intent(this, PhotoGalleryActivity.class), 0);

//通知信息

Notification notification = new NotificationCompat.Builder(this)

.setTicker(r.getString(R.string.new\_pictures\_title))

.setSmallIcon(android.R.drawable.ic\_menu\_report\_image)

.setContentTitle(r.getString(R.string.new\_pictures\_title))

.setContentText(r.getString(R.string.new\_pictures\_text))

.setContentIntent(pi)

.setAutoCancel(true)

.build();

NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION\_SERVICE);

notificationManager.notify(0, notification);

}

}

后台服务自动启动

//手机重启，应用程序不会重启，要让后台服务继续工作，实现一个广播接收者，需要监听手机启动事件（broadcast receiver, service, activity）

在配置文件中完成声明后，即使应用当前并未运行，只要有匹配的broadcast intent的发来，broadcast receiver就会接收，一收到intent,broadcast receiver的onReceive(Context, Intent)方法即开始运行，然后broadcast receiver就会被销毁

在AndroidManifest.xml

<uses-permission android:name="android.permission.RECEIVE\_BOOT\_COMPLETED" />

<receiver android:name=".StartupReceiver">

<intent-filter>

<action android:name="android.intent.action.BOOT\_COMPLETED" />

</intent-filter>

</receiver>

public class StartupReceiver extends BroadcastReceiver {

private static final String TAG = "StartupReceiver";

@Override public void onReceive(Context context, Intent intent) {

SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);

boolean isOn = prefs.getBoolean(PollService.PREF\_IS\_ALARM\_ON, false);

PollService.setServiceAlarm(context, isOn);

}

}

**广播信息，多个接收者处理**

//发送带权限的broadcast intent

在AndroidManifest.xml

<permission android:name="com.bignerdranch.android.photogallery.PRIVATE"

android:protectionLevel="signature" />

<uses-permission android:name="com.bignerdranch.android.photogallery.PRIVATE" />

Notification notification = …

Intent i = new Intent(ACTION\_SHOW\_NOTIFICATION);

i.putExtra("REQUEST\_CODE", requestCode);

i.putExtra("NOTIFICATION", notification);

sendOrderedBroadcast(i, PERM\_PRIVATE, null, null, Activity.RESULT\_OK, null, null);

//接收者1 （通过代码方式动态接收，注意权限PollService.PERM\_PRIVATE与AndroidManifest.xml设定一致）

在fragment,activity or 前端控制器

private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {

@Override public void onReceive(Context context, Intent intent) {

// 处理接收到的广播消息，并返回结果给广播者

setResultCode(Activity.RESULT\_CANCELED);

}

};

//注册与删除该activity监听广播信息

@Override public void onResume() {

super.onResume();

IntentFilter filter = new IntentFilter(PollService.ACTION\_SHOW\_NOTIFICATION);

getActivity().registerReceiver(mOnShowNotification, filter, PollService.PERM\_PRIVATE, null);

}

@Override public void onPause() {

super.onPause();

getActivity().unregisterReceiver(mOnShowNotification);

}

//接收者2 (设定为最后一个接收者，通过xml方式, 注意设定优先级为最小-999)

<receiver android:name=".NotificationReceiver" android:exported="false">

<intent-filter android:priority="-999">

<action android:name="com.bignerdranch.android.photogallery.SHOW\_NOTIFICATION" />

</intent-filter>

</receiver>

public class NotificationReceiver extends BroadcastReceiver {

private static final String TAG = "NotificationReceiver";

@Override public void onReceive(Context c, Intent i) {

if (getResultCode() != Activity.RESULT\_OK)

// a foreground activity cancelled the broadcast

return;

int requestCode = i.getIntExtra("REQUEST\_CODE", 0);

Notification notification = (Notification)i.getParcelableExtra("NOTIFICATION");

NotificationManager notificationManager = (NotificationManager)

c.getSystemService(Context.NOTIFICATION\_SERVICE);

notificationManager.notify(requestCode, notification);

}

}

## Android应用启动web:隐式Intent 和使用控件WebView

隐式Intent:

Uri photoPageUri = Uri.parse(urlString) ;

Intent I = new Intent(Intent.ACTION\_VIEW, photoPageUri) ;

startActivity(i) ;

使用控件WebView (见BAMA Android客户端)

Uri photoPageUri = Uri.parse(uriString);

Intent i = new Intent(getActivity(), PhotoPageActivity.class);

i.setData(photoPageUri);

startActivity(i);

在PhotoPageActivity包含的PhotoPageFragment中

@Override public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState) {

View v = inflater.inflate(R.layout.fragment\_photo\_page, parent, false);

final ProgressBar progressBar = (ProgressBar)v.findViewById(R.id.progressBar);

progressBar.setMax(100); // WebChromeClient reports in range 0-100

final TextView titleTextView = (TextView)v.findViewById(R.id.titleTextView);

mWebView = (WebView)v.findViewById(R.id.webView);

mWebView.getSettings().setJavaScriptEnabled(true);

mWebView.setWebViewClient(new WebViewClient() {

public boolean shouldOverrideUrlLoading(WebView view, String url) {

return false;

}

});

mWebView.setWebChromeClient(new WebChromeClient() {

public void onProgressChanged(WebView webView, int progress) {

if (progress == 100) {

progressBar.setVisibility(View.INVISIBLE);

} else {

progressBar.setVisibility(View.VISIBLE);

progressBar.setProgress(progress);

}

}

public void onReceivedTitle(WebView webView, String title) {

titleTextView.setText(title);

}

});

mWebView.loadUrl(mUrl);

return v;

}

//避免设备旋转时，WebView重新加载网页，解决方法是无需销毁activity，直接调整自己的视图以适应新的屏幕尺寸。Android:configChanges属性表明，如果因键盘开关、屏幕方向改变、屏幕大小改变而发生设备配置更改，那么activity应自己处理配置更改.

在AndroidManifest.xml

<activity android:name=".PhotoPageActivity"

android:configChanges="keyboardHidden|orientation|screenSize" />

## 自定义组件 (具体参考教材)

继承组件，覆盖方法

public class BoxDrawingView extends View {

// used when creating the view in code

public BoxDrawingView(Context context) {

this(context, null);

}

// used when inflating the view from XML

public BoxDrawingView(Context context, AttributeSet attrs) {

super(context, attrs);

}

@Override protected void onDraw(Canvas canvas) {

…

}

public boolean onTouchEvent(MotionEvent event) {

PointF curr = new PointF(event.getX(), event.getY());

switch (event.getAction()) {

case MotionEvent.ACTION\_DOWN:

// reset our drawing state

break;

case MotionEvent.ACTION\_MOVE:

if (mCurrentBox != null) {

mCurrentBox.setCurrent(curr);

invalidate();

}

break;

case MotionEvent.ACTION\_UP:

case MotionEvent.ACTION\_CANCEL:

mCurrentBox = null;

break;

}

return true;

}

}

//在布局文件中引用 (注意全路径)

<com.bignerdranch.android.draganddraw.BoxDrawingView

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

/>

## 获取设备地理位置信息

发送地理位置信息进行广播

String provider = LocationManager.GPS\_PROVIDER;

LocationManager mLocationManager =LocationManager)context.getSystemService(Context.LOCATION\_SERVICE);

Intent broadcast = new Intent(ACTION\_LOCATION);

PendingIntent pi = PendingIntent.getBroadcast(mAppContext, 0, broadcast, 0);

mLocationManager.requestLocationUpdates(provider, 0, 0, pi);

接收广播，获取地理位置信息

<receiver android:name=".LocationReceiver" android:exported="false">

<intent-filter>

<action android:name="com.bignerdranch.android.runtracker.ACTION\_LOCATION"/>

</intent-filter>

</receiver>

public class LocationReceiver extends BroadcastReceiver {

@Override public void onReceive(Context context, Intent intent) {

Location loc = (Location)intent.getParcelableExtra(LocationManager.KEY\_LOCATION\_CHANGED);

…

}

## SQLite本地数据库

Android提供SQLite数据库来处理这些数据，以磁盘上单个文件的形式存在，SQLite是一个开源的跨平台库，并具有一套强大的关系型数据库API可供使用。

设计应用的数据库存储API时，我们通常是为需要创建和管理的各类数据库创建一个SQLiteOpenHelper子类，大多应用只需要一个SQLiteOpenHelper子类，并与其余应用于组件共享一个类实例。

public class RunDatabaseHelper extends SQLiteOpenHelper {

private static final String DB\_NAME = "runs.sqlite";

private static final String TABLE\_RUN = "run";

private static final String COLUMN\_RUN\_ID = "\_id";

private static final String COLUMN\_RUN\_START\_DATE = "start\_date";

//设置数据库名

public RunDatabaseHelper(Context context) {

super(context, DB\_NAME, null, VERSION);

}

//创建数据库表

@Override public void onCreate(SQLiteDatabase db) {

// create the "run" table

db.execSQL("create table run (\_id integer primary key autoincrement, start\_date integer)");

}

//插入数据行

public long insertRun(Run run) {

ContentValues cv = new ContentValues();

cv.put(COLUMN\_RUN\_START\_DATE, run.getStartDate().getTime());

return getWritableDatabase().insert(TABLE\_RUN, null, cv);

}

//查询数据行,

public RunCursor queryRuns() {

// 查询条件：equivalent to "select \* from run order by start\_date asc"

Cursor wrapped = getReadableDatabase().query(TABLE\_RUN,

null, null, null, null, null, COLUMN\_RUN\_START\_DATE + " asc");

return new RunCursor(wrapped);

}

}

//通过游标获取数据行，cursor.moveToFirst(); cursor.isAfterLast();cursor.getRun(); cursor.close()

public static class RunCursor extends CursorWrapper {

public RunCursor(Cursor c) {super(c); }

public Run getRun() {

if (isBeforeFirst() || isAfterLast()) return null;

Run run = new Run();

run.setId(getLong(getColumnIndex(COLUMN\_RUN\_ID)));

run.setStartDate(new Date(getLong(getColumnIndex(COLUMN\_RUN\_START\_DATE))));

return run;

}

}

}

## 异步加载数据

现实开发中，我们应尽可能地避免在主线程上进行数据库操作。我们一般在后台线程上使用Loader,专门用于数据源加载某类数据（如对象），数据源可以是磁盘，数据库，ContentProvider,网络或者另一进程。Loader可在不阻塞主线程的情况下获取并发送结果数据给接收者.

每个Activity/Fragment都有一LoaderManager，管理着与loader间的所有通讯，并负责启动，停止和管理与组件关联的Loader生命周期方法。

//在RunListFragment.java中

Private RunListLoaderCallbacks runListLoaderCallback = new RunListLoaderCallbacks() ;

@Override public void onCreate(Bundle savedInstanceState) {

LoaderManager lm = getLoaderManager();

//初始化loader

lm.initLoader(LOAD\_RUN, args, runListLoaderCallback);

}

@Override public void onActivityResult(int requestCode, int resultCode, Intent data) {

if (REQUEST\_NEW\_RUN == requestCode) {

// restart the loader to get any new run available

getLoaderManager().restartLoader(0, null, runListLoaderCallback);

}

}

private class RunListLoaderCallbacks implements LoaderCallbacks<Cursor> {

@Override public Loader<Run> onCreateLoader(int id, Bundle args) {

return new RunListLoader(getActivity());

}

@Override public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {

// create an adapter to point at this cursor

RunCursorAdapter adapter = new RunCursorAdapter(getActivity(), (RunCursor)cursor);

//更新主线程

updateUI(adapter);

}

@Override public void onLoaderReset(Loader<Cursor> loader) {

updateUI(loader);

}

}

class RunListCursorLoader extends AsyncTaskLoader<Cursor> {

private Cursor mCursor ;

private long mRunId;

public RunListCursorLoader (Context context) {

super(context);

}

@Override public Cursor loadInBackground() {

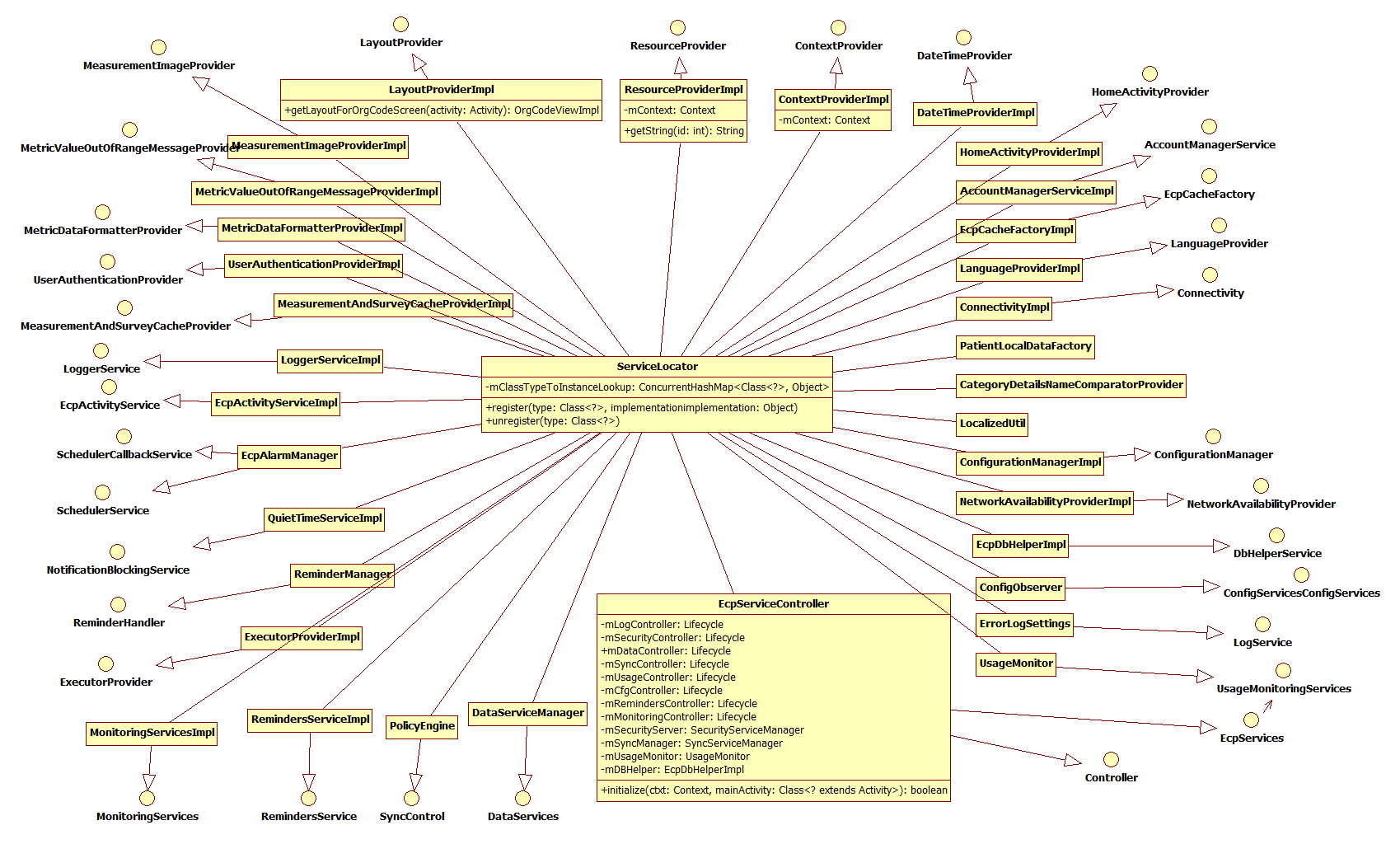
return RunManager.get(getContext()).queryRuns();

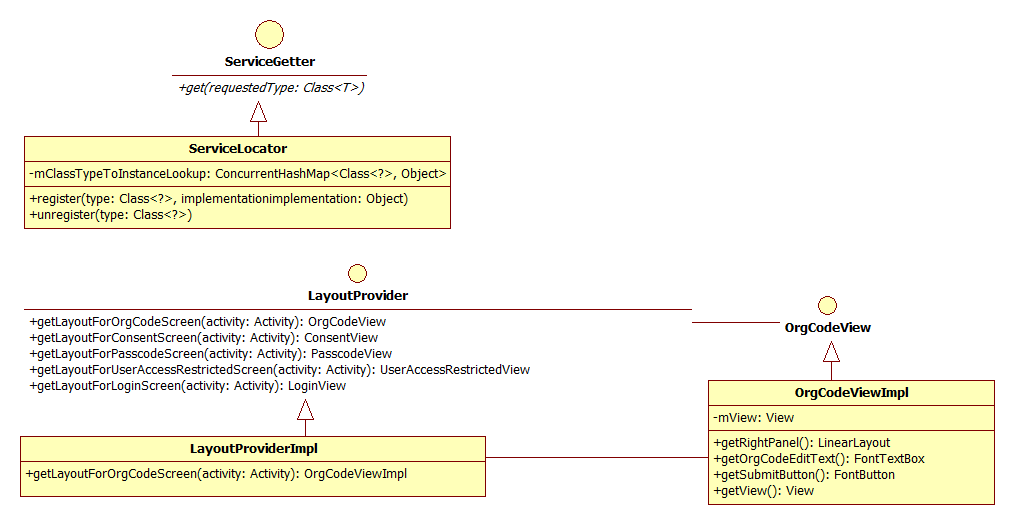
}

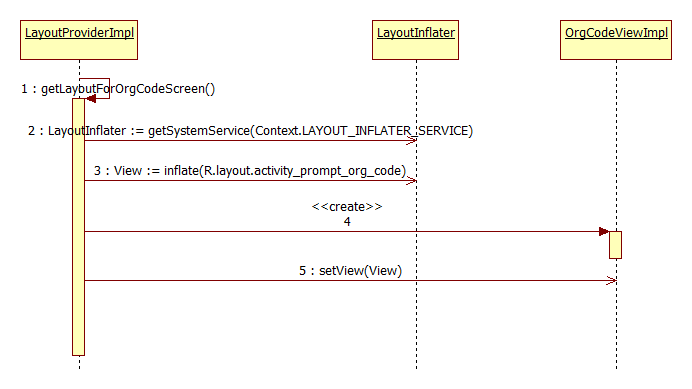
…

}

# <eCareCompanion>







1. EcpApplicaiton.java

@Override public void onCreate() {

ServiceLocator.*getInstance*().register(LayoutProvider.**class**, **new** LayoutProviderImpl());

ServiceLocator.*getInstance*().register(ResourceProvider.**class**, **new** ResourceProviderImpl(getApplicationContext()));

ServiceLocator.*getInstance*().register(ContextProvider.**class**, **new** ContextProviderImpl(getApplicationContext()));

ServiceLocator.*getInstance*().register(DateTimeProvider.**class**, **new** DateTimeProviderImpl());

ServiceLocator.*getInstance*().register(HomeActivityProvider.**class**, **new** HomeActivityProviderImpl(SplashScreenActivity.class));

ServiceLocator.*getInstance*().register(AccountManagerService.**class**, **new** AccountManagerServiceImpl());

ServiceLocator.getInstance().register(EcpCacheFactory.class, new EcpCacheFactoryImpl(mAppContext));

ServiceLocator.*getInstance*().register(LanguageProvider.**class**, **new** LanguageProviderImpl());

ServiceLocator.*getInstance*().register(Connectivity.**class**, **new** ConnectivityImpl());

EcpServiceController mController = new EcpServiceController();

ServiceLocator.getInstance().register(Controller.class, mController);

ServiceLocator.getInstance().register(EcpServices.class, mController);

ServiceLocator.getInstance().register(DbHelperService.class, new EcpDbHelperImpl());

EcpObserver observer = **new** EcpServicesObserver(networkAvailabilityProvider, getApplicationContext());

ServiceLocator.*getInstance*().get(EcpServices.**class**).registerObserver(observer);

ServiceLocator.getInstance().register(NetworkAvailabilityProvider.class, new NetworkAvailabilityProviderImpl());

ServiceLocator.*getInstance*().register(MeasurementImageProvider.**class**, **new** MeasurementImageProviderImpl());

ServiceLocator.*getInstance*().register(MetricValueOutOfRangeMessageProvider.**class**, **new** MetricValueOutOfRangeMessageProviderImpl());

ServiceLocator.*getInstance*().register(MetricDataFormatterProvider.**class**, **new** MetricDataFormatterProviderImpl());

ServiceLocator.*getInstance*().register(UserAuthenticationProvider.**class**, **new** UserAuthenticationProviderImpl()); ServiceLocator.*getInstance*().register(MeasurementAndSurveyCacheProvider.**class**, **new** MeasurementAndSurveyCacheProviderImpl());

ServiceLocator.getInstance().register(LoggerService.class, new LoggerServiceImpl());

ServiceLocator.getInstance().register(DataServices.class, new DataServiceManager());

