

82.5. Получение обновлений местоположения в приемнике BroadcastReceiver

Сначала создайте класс `BroadcastReceiver` для обработки входящих обновлений местоположения:

```
public class LocationReceiver extends BroadcastReceiver implements Constants {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (LocationResult.hasResult(intent)) {
            LocationResult locationResult = LocationResult.extractResult(intent);
            Location location = locationResult.getLastLocation();
            if (location != null) {
                // Сделать что-нибудь со своим местоположением
            } else {
                Log.d(LocationReceiver.class.getSimpleName(), "*** объект location является null ***");
            }
        }
    }
}
```

Затем при подключении к `GoogleApiClient` в обратном вызове `onConnected`:

```
@Override
public void onConnected(Bundle connectionHint) {
    Intent backgroundIntent = new Intent(this, LocationReceiver.class);
    mBackgroundPendingIntent = backgroundPendingIntent.
        getBroadcast(getApplicationContext(), LOCATION_REQUEST_CODE, backgroundIntent,
        PendingIntent.FLAG_CANCEL_CURRENT);
    mFusedLocationProviderApi.requestLocationUpdates(mLocationClient,
        mLocationRequest, backgroundPendingIntent);
}
```

Не забудьте удалить намерение обновления местоположения в соответствующем обратном вызове жизненного цикла:

```
@Override
public void onDestroy() {
    if (servicesAvailable && mLocationClient != null) {
        if (mLocationClient.isConnected()) {
            fusedLocationProviderApi.removeLocationUpdates(mLocationClient,
                backgroundPendingIntent);
            // Уничтожить клиента текущего местоположения
            mLocationClient = null;
        } else {
            mLocationClient.unregisterConnectionCallbacks(this);
            mLocationClient = null;
        }
    }
    super.onDestroy();
}
```

82.6. Регистрация GeoFence

Создадим синглтонный класс `GeoFenceObservationService`.

`GeoFenceObservationService.java`:

```
public class GeoFenceObservationService extends Service implements
    GoogleApiClient.ConnectionCallbacks, GoogleApiClient.OnConnectionFailedListener,
    ResultCallback<Status> {
```

```
protected static final String TAG = "GeoFenceObservationService";
protected GoogleApiClient mGoogleApiClient;
protected ArrayList<Geofence> mGeofenceList;
private boolean mGeofencesAdded;
private SharedPreferences mSharedPreferences;
private static GeoFenceObservationService mInstant;
public static GeoFenceObservationService getInstant(){
    return mInstant;
}

@Override
public void onCreate() {
    super.onCreate();
    mInstant = this;
    mGeofenceList = new ArrayList<Geofence>();
    mSharedPreferences = getSharedPreferences(AppConstants.SHARED_PREFERENCES_NAME, MODE_PRIVATE);
    mGeofencesAdded = mSharedPreferences.getBoolean(AppConstants.GEOFENCES_ADDED_KEY, false);
    buildGoogleApiClient();
}

@Override
public void onDestroy() {
    mGoogleApiClient.disconnect();
    super.onDestroy();
}

@Nullable
@Override
public IBinder onBind(Intent intent) {
    return null;
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    return START_STICKY;
}

protected void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
        .build();
    mGoogleApiClient.connect();
}

@Override
public void onConnected(Bundle connectionHint) {
}

@Override
public void onConnectionFailed(ConnectionResult result) {
}
```

```
@Override
public void onConnectionSuspended(int cause) {
}

private GeofencingRequest getGeofencingRequest() {

    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);
    builder.addGeofences(mGeofenceList);
    return builder.build();
}

public void addGeofences() {
    if (!mGoogleApiClient.isConnected()) {
        Toast.makeText(this, getString(R.string.not_connected), Toast.LENGTH_SHORT).show();
        return;
    }
    populateGeofenceList();
    if(!mGeofenceList.isEmpty()){
        try {
            LocationServices.GeofencingApi.addGeofences(mGoogleApiClient,
getGeofencingRequest(), getGeofencePendingIntent()).setResultCallback(this);
        } catch (SecurityException securityException) {
            securityException.printStackTrace();
        }
    }
}

public void removeGeofences() {
    if (!mGoogleApiClient.isConnected()) {
        Toast.makeText(this, getString(R.string.not_connected), Toast.LENGTH_SHORT).show();
        return;
    }
    try {
        LocationServices.GeofencingApi.
removeGeofences(mGoogleApiClient, getGeofencePendingIntent()).setResultCallback(this);
    } catch (SecurityException securityException) {
        securityException.printStackTrace();
    }
}

public void onResult(Status status) {

    if (status.isSuccess()) {
        mGeofencesAdded = !mGeofencesAdded;
        SharedPreferences.Editor editor = mSharedPreferences.edit();
        editor.putBoolean(AppConstants.GEOFENCES_ADDED_KEY, mGeofencesAdded);
    }
}
```

```
        editor.apply();
    } else {
        String errorMessage = AppConstants.getErrorString(this, status,
getStatusCode());
        Log.i("Geofence", errorMessage);
    }
}

private PendingIntent getGeofencePendingIntent() {
    Intent intent = new Intent(this, GeofenceTransitionsIntentService.class);
    return PendingIntent.getService(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
}

private void populateGeofenceList() {
    mGeofenceList.clear();
    List<GeoFencingResponce> geoFenceList = getGeofencesList;
    if(geoFenceList!=null&&!geoFenceList.isEmpty()){
        for (GeoFencingResponce obj : geoFenceList){
            mGeofenceList.add(obj.getGeofence());
            Log.i(TAG, "Registered Geofences : " + obj.Id+"-"+obj.Name+"-"+obj.
Latitude+"-"+obj.Longitude);
        }
    }
}
```

AppConstant:

```
public static final String SHARED_PREFERENCES_NAME = PACKAGE_NAME + ".SHARED_PREFERENCES_NAME";
public static final String GEOFENCES_ADDED_KEY = PACKAGE_NAME + ".GEOFENCES_ADDED_KEY";
public static final String DETECTED_GEOFENCES = "detected_geofences";
public static final String DETECTED_BEACONS = "detected_beacons";

public static String getErrorString(Context context, int errorCode) {
    Resources mResources = context.getResources();
    switch (errorCode) {
        case GeofenceStatusCodes.GEOFENCE_NOT_AVAILABLE:
            return mResources.getString(R.string.geofence_not_available);
        case GeofenceStatusCodes.GEOFENCE_TOO_MANY_GEOFENCES:
            return mResources.getString(R.string.geofence_too_many_geofences);
        case GeofenceStatusCodes.GEOFENCE_TOO_MANY_PENDING_INTENTS:
            return mResources.getString(R.string.geofence_too_many_pending_intents);
        default:
            return mResources.getString(R.string.unknown_geofence_error);
    }
}
```

Служба запускается из класса приложения:

- `startService(new Intent(getApplicationContext(), GeoFenceObservationService.class));`

Регистрация Geofences:

- `GeoFenceObservationService.getInstant().addGeofences();`

Глава 83. Тема, стиль, атрибут

83.1. Определение основного, основного темного и акцентного цветов

Вы можете настроить цветовую палитру своей темы (см. [#ColorPalette](https://developer.android.com/training/material/theme.html)) с помощью API фреймворка:

```
<style name="AppTheme" parent="Theme.Material">
    <item name="android:colorPrimary">@color/primary</item>
    <item name="android:colorPrimaryDark">@color/primary_dark</item>
    <item name="android:colorAccent">@color/accent</item>
</style>
```

...и с помощью библиотеки поддержки Appcompat (и AppCompatActivity):

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="colorPrimary">@color/primary</item>
    <item name="colorPrimaryDark">@color/primary_dark</item>
    <item name="colorAccent">@color/accent</item>
</style>
```

83.2. Несколько тем в одном приложении

Используя несколько тем в своем Android-приложении, можно добавить пользовательские цвета в каждую тему, чтобы получить примерно такой результат:



Для начала мы должны добавить наши темы в файл style.xml следующим образом:

```
<style name="OneTheme" parent="Theme.AppCompat.Light.DarkActionBar">
</style>

<!-- -->
<style name="TwoTheme" parent="Theme.AppCompat.Light.DarkActionBar" >
</style>
```

Выше представлены **OneTheme** и **TwoTheme**.

Теперь перейдите в файл `AndroidManifest.xml` и добавьте в тег `application` следующую строку: `android:theme="@style/OneTheme`, это сделает **OneTheme** темой по умолчанию:

```
<application
    android:theme="@style/OneTheme"
    ...>
```

Создайте новый xml-файл с именем `attrs.xml` и добавьте в него следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <attr name="custom_red" format="color" />
    <attr name="custom_blue" format="color" />
    <attr name="custom_green" format="color" />
</resources>
<!-- добавить все необходимые цвета (только название цвета) --&gt;</pre>

```

Вернитесь в `style.xml` и добавьте эти цвета с их значениями для каждой темы:

```
<style name="OneTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="custom_red">#8b030c</item>
    <item name="custom_blue">#0f1b8b</item>
    <item name="custom_green">#1c7806</item>
</style>

<style name="TwoTheme" parent="Theme.AppCompat.Light.DarkActionBar" >
    <item name="custom_red">#ff606b</item>
    <item name="custom_blue">#99cff</item>
    <item name="custom_green">#62e642</item>
</style>
```

Теперь у вас есть пользовательские цвета для каждой темы, давайте добавим эти цвета в наши представления. Добавьте цвет `custom_blue` в `TextView` с помощью `"?attr/"`:

Перейдите в `imageView` и добавьте этот цвет :

```
<TextView>
    android:id="@+id/txtte_view"
    android:textColor="?attr/custom_blue" />
```

Теперь мы можем изменить тему всего одной строкой `setTheme(R.style.TwoTheme);` эта строка должна находиться перед методом `setContentView()` в методе `onCreate()`, как в этом `Activity.java`:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setTheme(R.style.TwoTheme);
    setContentView(R.layout.main_activity);
    ...
}
```

Изменение темы для всех активностей одновременно

Если мы хотим изменить тему для всех активностей, нам необходимо создать новый класс с именем `MyActivity`, который расширяет класс `AppCompatActivity` (или класс `Activity`), и добавить в метод `onCreate()` строку `setTheme(R.style.TwoTheme);`

```
public class MyActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (new MySettings(this).isDarkTheme())
            setTheme(R.style.TwoTheme);
    }
}
```

Наконец, перейдите ко всем своим активностям и укажите, чтобы все они расширяли базовый класс `MyActivity`:

```
public class MainActivity extends MyActivity {
    ...
}
```

Для того чтобы изменить тему, достаточно зайти в `MyActivity` и изменить `R.style.TwoTheme` на свою тему (`R.style.OneTheme`, `R.style.ThreeTheme`).

83.3. Цвет панели навигации

Этот атрибут используется для изменения панели навигации.

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="android:navigationBarColor">@color/my_color</item>
</style>
```

83.4. Использование пользовательской темы для каждой активности

В файле `themes.xml`:

```
<style name="MyActivityTheme" parent="Theme.AppCompat">
    <!-- Атрибуты темы здесь -->
</style>
```

В файле `AndroidManifest.xml`:

```
<application
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/Theme.AppCompat">

    <activity
        android:name=".MyActivity"
        android:theme="@style/MyActivityTheme" />

</application>
```

83.5. Глобальное использование пользовательской темы

В файле `themes.xml`:

```
<style name="AppTheme" parent="Theme.AppCompat">
    <!-- Атрибуты темы здесь -->
</style>
```

В файле AndroidManifest.xml:

```
<application
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <!-- Объявления активности здесь -->
</application>
```

83.6. Цвет прокрутки

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="android:colorEdgeEffect">@color/my_color</item>
</style>
```

83.7. Цвет пульсации

Анимация пульсации отображается при нажатии пользователем на кликабельные виды. Вы можете использовать тот же цвет пульсации, что и в вашем приложении, присваивая атрибут ?android:colorControlHighlight в ваших представлениях. Можно настроить этот цвет, изменив атрибут android:colorControlHighlight в своей теме. Цвет этого эффекта может быть изменен:

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="android:colorControlHighlight">@color/my_color</item>
</style>
```

Или, если вы используете тему Material Theme:

```
<style name="AppTheme" parent="android:Theme.Material.Light">
    <item name="android:colorControlHighlight">@color/your_custom_color</item>
</style>
```

83.8. Полупрозрачные панели навигации и состояния

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="android:windowTranslucentNavigation">true</item>
</style>
```

Строчку состояния можно сделать прозрачной, применив этот атрибут к стилю:

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="android:windowTranslucentStatus">true</item>
</style>
```

83.9. Наследование тем

При определении тем обычно используется тема, предоставляемая системой, а затем изменяется ее внешний вид в соответствии с приложением. Например, так наследуется тема Theme.AppCompat:

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="colorPrimary">@color/colorPrimary</item>
```

```
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
</style>
```

Теперь эта тема имеет все свойства стандартной темы `Theme.AppCompat`, кроме тех, которые мы изменили явно.

Существует также ярлык при наследовании, который обычно используется при наследовании от собственной темы:

```
<style name="AppTheme.Red">
    <item name="colorAccent">@color/red</item>
</style>
```

Поскольку в начале его имени уже стоит `AppTheme.`, он автоматически наследует его, без необходимости определять родительскую тему. Это удобно, когда необходимо создать определенные стили для части приложения (например, для одной активности).

Глава 84. MediaPlayer

84.1. Базовое создание и воспроизведение

Класс `MediaPlayer` может использоваться для управления воспроизведением файлов и потоков аудио/видео. Создание объекта `MediaPlayer` может быть трех типов.

1. Медиа с локального ресурса:

```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.resource);
mediaPlayer.start();
// нет необходимости вызывать prepare(); create() сделает это за вас
```

2. Из локального URI (полученного от `ContentResolver`):

```
Uri myUri = ....; // инициализация Uri здесь
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(getApplicationContext(), myUri);
mediaPlayer.prepare();
mediaPlayer.start();
```

3. Из внешнего URL:

```
String url = "http://....."; // здесь ваш URL
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(url);
mediaPlayer.prepare(); // может занять много времени! (на буферизацию и т.д.)
mediaPlayer.start();
```

84.2. Медиаплеер с прогрессом буфера и позицией воспроизведения

```
public class SoundActivity extends Activity {
    private MediaPlayer mediaPlayer;
    ProgressBar progress_bar;
```

```
    @Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_tool_sound);
    mediaPlayer = new MediaPlayer();
    mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
    progress_bar = (ProgressBar) findViewById(R.id.progress_bar);

    btn_play_stop.setEnabled(false);
    btn_play_stop.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if(mediaPlayer.isPlaying()) {
                mediaPlayer.pause();
                btn_play_stop.setImageResource(R.drawable.ic_pause_black_24dp);
            } else {
                mediaPlayer.start();
                btn_play_stop.setImageResource(R.drawable.ic_play_arrow_
black_24px);
            }
        }
    });

    mediaPlayer.setDataSource(proxyUrl);
    mediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener()
{
    @Override
    public void onCompletion(MediaPlayer mp) {
        observer.stop();
        progress_bar.setProgress(mp.getCurrentPosition());
        // TODO Автоматически сгенерированная заглушка метода
        mediaPlayer.stop();
        mediaPlayer.reset();
    }
});
    mediaPlayer.setOnBufferingUpdateListener(new MediaPlayer.
OnBufferingUpdateListener() {
        @Override
        public void onBufferingUpdate(MediaPlayer mp, int percent) {
            progress_bar.setSecondaryProgress(percent);
        }
});
    mediaPlayer.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
        @Override
        public void onPrepared(MediaPlayer mediaPlayer) {
            btn_play_stop.setEnabled(true);
        }
});
    observer = new MediaObserver();
    mediaPlayer.prepare();
    mediaPlayer.start();
    new Thread(observer).start();
}

private MediaObserver observer = null;

private class MediaObserver implements Runnable {
    private AtomicBoolean stop = new AtomicBoolean(false);
}
```

```
public void stop() {
    stop.set(true);
}

@Override
public void run() {
    while (!stop.get()) {
        progress_bar.setProgress((int)((double)mediaPlayer.
getCursorPosition() / (double)mediaPlayer.getDuration()*100));
        try {
            Thread.sleep(200);
        } catch (Exception ex) {
            Logger.log(ToolSoundActivity.this, ex);
        }
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    mediaPlayer.stop();
}
```

```
<LinearLayout
    android:gravity="bottom"
    android:layout_gravity="bottom"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:weightSum="1">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <ImageButton
            app:srcCompat="@drawable/ic_play_arrow_black_24px"
            android:layout_width="48dp"
            android:layout_height="48dp"
            android:id="@+id	btn_play_stop" />

        <ProgressBar
            android:padding="8dp"
            android:progress="0"
            android:id="@+id/progress_bar"
            style="@style/Widget.AppCompat.ProgressBar.Horizontal"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center" />

    </LinearLayout>
</LinearLayout>
```

84.3. Получение системных мелодий

В данном примере показано, как получить Uri системных мелодий звонка (RingtoneManager.TYPE_RINGTONE):

```
private List<Uri> loadLocalRingtonesUris() {
    List<Uri> alarms = new ArrayList<>();
    try {
        RingtoneManager ringtoneMgr = new RingtoneManager(getActivity());
        ringtoneMgr.setType(RingtoneManager.TYPE_RINGTONE);

        Cursor alarmsCursor = ringtoneMgr.getCursor();
        int alarmsCount = alarmsCursor.getCount();
        if (alarmsCount == 0 && !alarmsCursor.moveToFirst()) {
            alarmsCursor.close();
            return null;
        }

        while (!alarmsCursor.isAfterLast() && alarmsCursor.moveToNext()) {
            int currentPosition = alarmsCursor.getPosition();
            alarms.add(ringtoneMgr.getRingtoneUri(currentPosition));
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    return alarms;
}
```

Список зависит от типов запрашиваемых мелодий. Возможны следующие варианты:

- RingtoneManager.TYPE_RINGTONE
- RingtoneManager.TYPE_NOTIFICATION
- RingtoneManager.TYPE_ALARM
- RingtoneManager.TYPE_ALL = TYPE_RINGTONE | TYPE_NOTIFICATION | TYPE_ALARM

Для получения мелодий в формате android.media.Ringtone каждый Uri должен быть разрешен RingtoneManager:

```
android.media.Ringtone osRingtone = RingtoneManager.getRingtone(context, uri);
```

Для воспроизведения звука используйте метод:

```
public void setDataSource(Context context, Uri uri)
```

...из android.media.MediaPlayer. MediaPlayer должен быть инициализирован и подготовлен в соответствии с диаграммой состояний (State diagram, см. <https://developer.android.com/reference/android/media/MediaPlayer.html#StateDiagram>).

84.4. Асинхронная подготовка

Вызов MediaPlayer\$prepare() является блокирующим и приводит к зависанию пользовательского интерфейса до завершения выполнения. Чтобы решить эту проблему, можно использовать MediaPlayer\$prepareAsync().

```
mMediaPlayer = ... // Инициализируем его здесь
mMediaPlayer.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
    @Override
    public void onPrepared(MediaPlayer player) {
```

```

    // Вызывается, когда медиаплеер готов к воспроизведению
    mMediaPlayer.start();
}

}); // Установка обратного вызова для момента завершения работы prepareAsync()
mMediaPlayer.prepareAsync(); // Подготовка выполняется асинхронно, чтобы
не блокировать основной поток

```

При синхронных операциях ошибки обычно сигнализируются исключением или кодом ошибки, но при использовании асинхронных ресурсов необходимо обеспечить соответствующее уведомление приложения об ошибках. Для MediaPlayer:

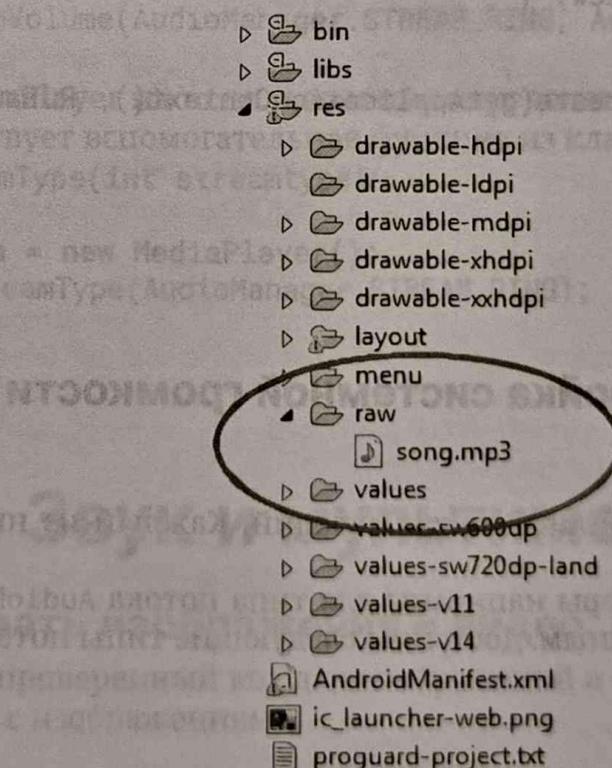
```

mMediaPlayer.setOnErrorListener(new MediaPlayer.OnErrorListener() {
    @Override
    public boolean onError(MediaPlayer mp, int what, int extra) {
        // ... реагировать соответствующим образом ...
        // Медиаплеер перешел в состояние Error, необходим сброс!
        // Затем вернуть true, если ошибка была обработана
    }
});

```

84.5. Импорт звука в Android Studio и его воспроизведение

Приведем пример того, как получить возможность воспроизведения аудиофайла, который уже есть на вашем ПК или ноутбуке. Сначала создайте новый подкаталог в res и назовите его raw следующим образом:



Скопируйте звук, который вы хотите воспроизвести, в эту папку. Это может быть файл mp3 или .wav.

К примеру, при нажатии на кнопку вы хотите воспроизвести этот звук, вот как это делается:

```

public class MainActivity extends AppCompatActivity {
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.aboutapp_activity);
}

```

```

MediaPlayer song=MediaPlayer.create(this, R.raw.song);

Button button=(Button)findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        song.start();
    }
});
}
}

```

При этом песня будет проигрываться только один раз при нажатии на кнопку, если вы хотите проигрывать песню при каждом нажатии на кнопку, используйте следующий код:

```

public class MainActivity extends AppCompatActivity {
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.aboutapp_activity);

    MediaPlayer song=MediaPlayer.create(this, R.raw.song);

    Button button=(Button)findViewById(R.id.button);
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if (song.isPlaying()) {
                song.reset();
                song= MediaPlayer.create(getApplicationContext(), R.raw.song);
            }
            song.start();
        }
    });
}
}

```

84.6. Получение и настройка системной громкости

Типы аудиопотоков

Существуют различные профили потоков мелодий. Каждый из них имеет свою громкость.

Все приведенные здесь примеры написаны для типа потока `AudioManager.STREAM_RING`. Однако он не является единственным. Доступны следующие типы потоков:

- `STREAM_ALARM`
- `STREAM_DTMF`
- `STREAM_MUSIC`
- `STREAM_NOTIFICATION`
- `STREAM_RING`
- `STREAM_SYSTEM`
- `STREAM_VOICE_CALL`

Установка громкости

Чтобы узнать громкость конкретного профиля, вызовите:

```
AudioManager audio = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
```

```
int currentVolume = audioManager.getStreamVolume(AudioManager.STREAM_RING);
```

Это значение приносит мало пользы, если максимальное значение для потока неизвестно:

```
AudioManager audio = (AudioManager) getActivity().getSystemService(Context.AUDIO_SERVICE);
int streamMaxVolume = audioManager.getStreamMaxVolume(AudioManager.STREAM_RING);
```

Отношение этих двух значений даст относительную громкость (relative volume) ($0 <$ громкость < 1):

```
float volume = ((float) currentVolume) / streamMaxVolume
```

Регулировка громкости на один шаг

Чтобы увеличить громкость потока на один шаг, вызовите:

```
AudioManager audio = (AudioManager) getActivity().getSystemService(Context.AUDIO_SERVICE);
audio.adjustStreamVolume(AudioManager.STREAM_RING, AudioManager.ADJUST_RAISE, 0);
```

Чтобы уменьшить громкость потока на одну ступень, вызовите:

```
AudioManager audio = (AudioManager) getActivity().getSystemService(Context.AUDIO_SERVICE);
audio.adjustStreamVolume(AudioManager.STREAM_RING, AudioManager.ADJUST_LOWER, 0);
```

Настройка MediaPlayer на использование определенного типа потока

Для этого существует вспомогательная функция из класса MediaPlayer. Просто вызовите `void setAudioStreamType(int streamtype)`:

```
MediaPlayer mMedia = new MediaPlayer();
mMedia.setAudioStreamType(AudioManager.STREAM_RING);
```

Глава 85. Звук и мультимедиа в Android

85.1. Как выбрать изображение и видео

Ниже приведен проверенный код для изображений и видео. Он будет работать для всех API выше 19. Работа с изображениями:

```
Intent intent = new Intent(Intent.ACTION_PICK, android.provider.MediaStore.Images.
Media.EXTERNAL_CONTENT_URI);
startActivityForResult(intent, 10);
```

Работа с видео:

```
Intent intent = new Intent(Intent.ACTION_PICK, android.provider.MediaStore.Video.
Media.EXTERNAL_CONTENT_URI);
startActivityForResult(intent, 20);

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

```

super.onActivityResult(requestCode, resultCode, data);
if (resultCode == Activity.RESULT_OK) {

    if (requestCode == 10) {
        Uri selectedImageUri = data.getData();
        String selectedImagePath = getRealPathFromURI(selectedImageUri);
    } else if (requestCode == 20) {
        Uri selectedVideoUri = data.getData();
        String selectedVideoPath = getRealPathFromURI(selectedVideoUri);
    }
}

public String getRealPathFromURI(Uri uri) {
    if (uri == null) {
        return null;
    }
    String[] projection = {MediaStore.Images.Media.DATA};
    Cursor cursor = getActivity().getContentResolver().query(uri, projection,
        null, null, null);
    if (cursor != null) {
        int column_index = cursor
            .getColumnIndexOrThrow(MediaStore.Images.Media.DATA);
        cursor.moveToFirst();
        return cursor.getString(column_index);
    }
    return uri.getPath();
}

```

85.2. Воспроизведение звука через SoundPool

```

public class PlaySound extends Activity implements OnTouchListener {
    private SoundPool soundPool;
    private int soundID;
    boolean loaded = false;

    /** Вызывается при первом создании активности. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        View view = findViewById(R.id.textView1); view.setOnTouchListener(this);
        // Установка аппаратных кнопок для управления музыкой
        this.setVolumeControlStream(AudioManager.STREAM_MUSIC);
        // Загрузка звука
        soundPool = new SoundPool(10, AudioManager.STREAM_MUSIC, 0);
        soundPool.setOnLoadCompleteListener(new OnLoadCompleteListener() {
            @Override
            public void onLoadComplete(SoundPool soundPool, int sampleId, int
                status) {
                loaded = true;
            }
        });
        soundID = soundPool.load(this, R.raw.sound1, 1);

    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {

```

```

// Получение настроек звука пользователя
AudioManager audioManager = (AudioManager) getSystemService(AUDIO_SERVICE);
float actualVolume = (float) audioManager
    .getStreamVolume(AudioManager.STREAM_MUSIC);
float maxVolume = (float) audioManager
    .getStreamMaxVolume(AudioManager.STREAM_MUSIC);
float volume = actualVolume / maxVolume;
// Звук уже загружен?
if (loaded) {
    soundPool.play(soundID, volume, volume, 1, 0, 1f);
    Log.e("Test", "Played sound");
}
}
return false;
}
}

```

Глава 86. MediaSession

86.1. Получение и обработка событий от кнопок

В данном примере при запуске службы создается объект `MediaSession`. Он освобождается при уничтожении службы:

```

public final class MyService extends Service {
    private static MediaSession s_mediaSession;

    @Override
    public void onCreate() {
        // Инстанцируем новый объект MediaSession
        configureMediaSession();
    }

    @Override
    public void onDestroy() {
        if (s_mediaSession != null)
            s_mediaSession.release();
    }
}

```

Следующий метод инстанцирует и настраивает обратные вызовы кнопок `MediaSession`:

```

private void configureMediaSession {
    s_mediaSession = new MediaSession(this, "MyMediaSession");

    // Переопределенные методы в классе MediaSession.Callback
    s_mediaSession.setCallback(new MediaSession.Callback() {
        @Override
        public boolean onMediaButtonEvent(Intent mediaButtonIntent) {
            Log.d(TAG, "onMediaButtonEvent called: " + mediaButtonIntent);
            KeyEvent ke = mediaButtonIntent.getParcelableExtra(Intent.EXTRA_KEY_EVENT);
            if (ke != null && ke.getAction() == KeyEvent.ACTION_DOWN) {

```

```

        int KeyCode = ke.getKeyCode();
        Log.d(TAG, "onMediaButtonEvent Получена команда: " + ke);
    }
    return super.onMediaButtonEvent(mediaButtonIntent);
}

@Override
public void onSkipToNext() {
    Log.d(TAG, "onSkipToNext called (media button pressed)");
    Toast.makeText(getApplicationContext(), "onSkipToNext called", Toast.LENGTH_SHORT).show();
    skipToNextPlaylistItem(); // Обработка нажатия этой кнопки
    super.onSkipToNext();
}

@Override
public void onSkipToPrevious() {
    Log.d(TAG, "onSkipToPrevious called (media button pressed)");
    Toast.makeText(getApplicationContext(), "onSkipToPrevious called", Toast.LENGTH_SHORT).show();
    skipToPreviousPlaylistItem(); // Обработка нажатия этой кнопки
    super.onSkipToPrevious();
}

@Override
public void onPause() {
    Log.d(TAG, "onPause called (media button pressed)");
    Toast.makeText(getApplicationContext(), "onPause called", Toast.LENGTH_SHORT).show();
    mpPause(); // Приостановка проигрывателя
    super.onPause();
}

@Override
public void onPlay() {
    Log.d(TAG, "onPlay called (media button pressed)");
    mpStart(); // Запуск плеера/воспроизведения.
    super.onPlay();
}

@Override
public void onStop() {
    Log.d(TAG, "onStop called (media button pressed)");
    mpReset(); // Остановка и/или сброс проигрывателя.
    super.onStop();
}
});

s_mediaSession.setFlags(MediaSession.FLAG_HANDLES_MEDIA_BUTTONS | MediaSession.FLAG_HANDLES_TRANSPORT_CONTROLS);
s_mediaSession.setActive(true);
}

```

Следующий метод отправляет метаданные (хранящиеся в `HashMap`) на устройство с помощью A2DP:

```

void sendMetaData(@NonNull final HashMap<String, String> hm) {
    // Возврат, если Bluetooth A2DP не используется.
    if (!((AudioManager) getSystemService(Context.AUDIO_SERVICE)).
        isBluetoothA2dpOn()) return;
}

```

```

MediaMetadata metadata = new MediaMetadata.Builder()
    .putString(MediaMetadata.METADATA_KEY_TITLE, hm.get("Title"))
    .putString(MediaMetadata.METADATA_KEY_ALBUM, hm.get("Album"))
    .putString(MediaMetadata.METADATA_KEY_ARTIST, hm.get("Artist"))
    .putString(MediaMetadata.METADATA_KEY_AUTHOR, hm.get("Автор"))
    .putString(MediaMetadata.METADATA_KEY_COMPOSER, hm.get("Composer"))
    .putString(MediaMetadata.METADATA_KEY_WRITER, hm.get("Writer"))
    .putString(MediaMetadata.METADATA_KEY_DATE, hm.get("Date"))
    .putString(MediaMetadata.METADATA_KEY_GENRE, hm.get("Genre"))
    .putLong(MediaMetadata.METADATA_KEY_YEAR, tryParse(hm.get("Year")))
    .putLong(MediaMetadata.METADATA_KEY_DURATION, tryParse(hm.get("Raw
Duration")))
    .putLong(MediaMetadata.METADATA_KEY_TRACK_NUMBER, tryParse(hm.get("Track
Number")))
    .build();

s_mediaSession.setMetadata(metadata);
}

```

Следующий метод устанавливает состояние воспроизведения (`PlaybackState`, см. о нем <https://developer.android.com/reference/android/media/session/PlaybackState.html>). Он также определяет, на какие действия кнопок будет реагировать `MediaSession`:

```

private void setPlaybackState(@NonNull final int stateValue) {
    PlaybackState state = new PlaybackState.Builder()
        .setActions(PlaybackState.ACTION_PLAY | PlaybackState.ACTION_SKIP_TO_NEXT
            | PlaybackState.ACTION_PAUSE | PlaybackState.ACTION_SKIP_TO_PREVIOUS
            | PlaybackState.ACTION_STOP | PlaybackState.ACTION_PLAY_PAUSE)
        .setState(stateValue, PlaybackState.PLAYBACK_POSITION_UNKNOWN, 0)
        .build();
    s_mediaSession.setPlaybackState(state);
}

```

Глава 87. MediaStore

87.1. Выборка аудио/MP3-файлов из определенной папки устройства или выборка всех файлов

Сначала добавьте в манифест проекта следующие разрешения, чтобы разрешить доступ к хранилищу устройства:

```

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

```

Затем создайте файл `AudioModel.class` и поместите в него следующий класс модели, позволяющий получать и устанавливать элементы списка:

```

public class AudioModel {
    String aPath;
    String aName;
    String aAlbum;
    String aArtist;

    public String getaPath() {
        return aPath;
    }
}

```

```

public void setaPath(String aPath) {
    this.aPath = aPath;
}
public String getaName() {
    return aName;
}
public void setaName(String aName) {
    this.aName = aName;
}
public String getaAlbum() {
    return aAlbum;
}
public void setaAlbum(String aAlbum) {
    this.aAlbum = aAlbum;
}
public String getaArtist() {
    return aArtist;
}
public void setaArtist(String aArtist) {
    this.aArtist = aArtist;
}
}

```

Далее используйте следующий метод для чтения всех MP3-файлов из папки устройства или для чтения всех файлов устройства:

```

public List<AudioModel> getAllAudioFromDevice(final Context context) {
    final List<AudioModel> tempAudioList = new ArrayList<>();

    Uri uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
    String[] projection = {MediaStore.Audio.AudioColumns.DATA, MediaStore.Audio.
        AudioColumns.TITLE, MediaStore.Audio.AudioColumns.ALBUM, MediaStore.Audio.
        ArtistColumns.ARTIST,};
    Cursor c = context.getContentResolver().query(uri, projection, MediaStore.Audio.
        Media.DATA + " like ? ", new String[]{"%utm%"}, null);

    if (c != null) {
        while (c.moveToFirst()) {
            AudioModel audioModel = new AudioModel();
            String path = c.getString(0);
            String name = c.getString(1);
            String album = c.getString(2);
            String artist = c.getString(3);

            audioModel.setaName(name);
            audioModel.setaAlbum(album);
            audioModel.setaArtist(artist);
            audioModel.setaPath(path);

            Log.e("Name :" + name, " Album :" + album);
            Log.e("Path :" + path, " Artist :" + artist);

            tempAudioList.add(audioModel);
        }
        c.close();
    }

    return tempAudioList;
}

```

Приведенный выше код вернет список всех MP3-файлов с указанием названия, пути, исполнителя и альбома. Для получения более подробной информации обратитесь к документации по `Media.Store.Audio` (<https://developer.android.com/reference/android/provider/MediaStore.Audio.html>).

Для того чтобы прочитать файлы конкретной папки, используйте следующий запрос (имя папки необходимо заменить):

```
Cursor c = context.getContentResolver().query(uri,
    projection,
    MediaStore.Audio.Media.DATA + " like ?",
    new String[]{"%yourFolderName%"}, // Укажите здесь местоположение папки/файла
    null);
```

Если необходимо получить все файлы с устройства, то используйте следующий запрос:

```
Cursor c = context.getContentResolver().query(uri,
    projection,
    null,
    null,
    null);
```

Примечание: не забудьте включить разрешения на доступ к хранилищу.

Теперь для получения MP3-файлов достаточно вызвать описанный выше метод:

```
getAllAudioFromDevice(this);
```

Пример с активностью

```
public class ReadAudioFilesActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_audio_list);

        /**
         * В результате будет получен список всех MP3-файлов. Используйте список для
         * отображения данных.
         */
        getAllAudioFromDevice(this);
    }
    // Метод для чтения всех аудио/MP3 файлов.
    public List<AudioModel> getAllAudioFromDevice(final Context context) {
        final List<AudioModel> tempAudioList = new ArrayList<>();

        Uri uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
        String[] projection =
        {MediaStore.Audio.AudioColumns.DATA, MediaStore.Audio.AudioColumns.TITLE,
        MediaStore.Audio.AudioColumns.ALBUM, MediaStore.Audio.ArtistColumns.ARTIST,};

        Cursor c = context.getContentResolver().query(uri, projection, MediaStore.
        Audio.Media.DATA + " like ? ", new String[]{"%utm%"}, null);

        if (c != null) {
            while (c.moveToNext()) {
                // Создание объекта модели.
                AudioModel audioModel = new AudioModel();
                ...
            }
        }
    }
}
```

```

        String path = c.getString(0); // Получение пути.
        String name = c.getString(1); // Получение названия.
        String album = c.getString(2); // Получение названия альбома.
        String artist = c.getString(3); // Получение имени исполнителя.

        // Устанавливаем данные в объект модели.
        audioModel.setaName(name);
        audioModel.setaAlbum(album);
        audioModel.setaArtist(artist);
        audioModel.setaPath(path);

        Log.e("Name :" + name, " Album :" + album);
        Log.e("Path :" + path, " Artist :" + artist);

        // Добавить объект модели в список.
        tempAudioList.add(audioModel);
    }
    c.close();
}

// Возвращаем список.
return tempAudioList;
}
}

```

Глава 88. Multidex и предел метода Dex

Под DEX понимаются исполняемые файлы байт-кода приложения Android (APK) в виде файлов Dalvik Executable (DEX), которые содержат скомпилированный код, используемый для запуска вашего приложения.

Спецификация Dalvik Executable ограничивает общее количество методов, на которые можно ссылаться в одном DEX-файле, до 65 536 (64К) – включая методы фреймворка Android, библиотечные методы и методы в собственном коде.

Для преодоления этого ограничения необходимо настроить процесс сборки приложения на генерацию более чем одного DEX-файла, называемого Multidex.

88.1. Включение Multidex

Для того чтобы включить конфигурацию Multidex, необходимо:

- изменить конфигурацию сборки Gradle
- использовать MultiDexApplication или включить MultiDex в класс приложения

Конфигурация Gradle

В app/build.gradle добавьте эти части:

```

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.1"

    defaultConfig {
        ...
        minSdkVersion 14
    }
}

```

```

targetSdkVersion 24
...
// Включение поддержки MultiDex
multiDexEnabled true
}
...
}
mainContent() // или toolbar
}
dependencies {
    compile 'com.android.support:multidex:1.0.1'
}
classpath 'com.getkeepsafe.dexcount.gradle.plugin:plugin:1.0.0'

```

Включение MultiDex в приложение

Используйте один из трех вариантов реализации:

- Multidex путем расширения приложения
- Multidex путем расширения MultiDexApplication
- Multidex с использованием MultiDexApplication напрямую

При добавлении этих конфигурационных параметров в приложение средства сборки Android создают основной файл dex (classes.dex) и вспомогательные файлы (classes2.dex, classes3.dex) по мере необходимости.

Затем система сборки упакует их в APK-файл для распространения.

88.2. Реализация Multidex путем расширения приложения

Используйте эту опцию, если в проекте требуется подкласс Application.

Укажите этот подкласс Application с помощью свойства android:name в файле манифеста внутри тега application. В подкласс Application добавьте переопределение метода attachBaseContext() и в этом методе вызовите MultiDex.install():

```

пакет com.example;

import android.app.Application;
import android.content.Context;

/**
 * Расширенное приложение, поддерживающее мультидекс
 */
public class MyApplication extends Application {

    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        MultiDex.install(this);
    }
}

```

Убедитесь, что в файле AndroidManifest.xml в теге application указан подкласс Application:

```

<application
    android:name="com.example.MyApplication"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name">
</application>

```

88.3. Реализация Multidex путем расширения MultiDexApplication

Этот способ очень похож на использование подкласса Application и переопределение метода attachBaseContext(). Однако при использовании этого способа нет необходимости переопределять attachBaseContext(), так как это уже сделано в суперклассе MultiDexApplication.

Расширим MultiDexApplication вместо Application:

```
package com.example;

import android.support.multidex.MultiDexApplication;
import android.content.Context;

/**
 * Расширенное приложение MultiDexApplication
 */
public class MyApplication extends MultiDexApplication {

    // Нет необходимости переопределять attachBaseContext()
    //.....
}
```

Добавьте этот класс в *AndroidManifest.xml* точно так же, как если бы вы расширяли Application:

```
<application
    android:name="com.example.MyApplication"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name">
</application>
```

88.4. Реализация Multidex при использовании MultiDexApplication напрямую

Используйте этот способ, если вам не нужен подкласс Application.

Это самый простой вариант, но в этом случае вы не сможете предоставить свой собственный подкласс Application. Если подкласс нужен, то придется перейти к одному из других вариантов.

Для этого способа достаточно указать полное имя класса android.support.multidex.MultiDexApplication для свойства android:name тега application в файле *AndroidManifest.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.multidex.myapplication">
    <application
        ...
        android:name="android.support.multidex.MultiDexApplication">
        ...
    </application>
</manifest>
```

88.5. Подсчет ссылок на методы при каждой сборке (плагин Dexcount Gradle)

Плагин dexcount (см. <https://github.com/KeepSafe/dexcount-gradle-plugin>) подсчитывает количество методов и ресурсов классов после успешной сборки.

Добавьте плагин в app/build.gradle:

```
apply plugin: 'com.android.application'

buildscript {
    repositories {
        mavenCentral() // или jcenter()
    }
    dependencies {
        classpath 'com.getkeepsafe.dexcount:dexcount-gradle-plugin:0.5.5'
    }
}
```

Примените плагин в файле app/build.gradle:

```
apply plugin: 'com.getkeepsafe.dexcount'
```

Найдите выходные данные, сгенерированные плагином, в:

..../app/build/outputs/dexcount

Особенно полезен debugChart в формате .html, расположенный в:

..../app/build/outputs/dexcount/debugChart/index.html

Глава 89. Синхронизация данных с помощью Sync Adapter

89.1. Фиктивный адаптер синхронизации с провайдером-заглушкой

SyncAdapter:

```
/**
 * Определите адаптер синхронизации для приложения.
 * Этот класс инстанцируется в {@link SyncService}, который также связывает
 * SyncAdapter с системой.
 * SyncAdapter должен инициализироваться только в SyncService, и нигде больше.
 * Система вызывает onPerformSync() посредством RPC-вызыва через объект IBinder,
 * предоставленный SyncService.
 */

class SyncAdapter extends AbstractThreadingSyncAdapter {
    /**
     * Конструктор. Получает дескриптор к распознавателю содержимого для последующего
     * использования.
     */
    public SyncAdapter(Context context, boolean autoInitialize) {
        super(context, autoInitialize);
    }
}
```

```

/**
 * Конструктор. Получает дескриптор к распознавателю содержимого для последующего
 * использования.
 */
public SyncAdapter(Context context, boolean autoInitialize, boolean
allowParallelSyncs) {
    super(context, autoInitialize, allowParallelSyncs);
}

@Override
public void onPerformSync(Account account, Bundle extras, String authority,
ContentProviderClient provider, SyncResult syncResult) {
//Задания, которые вы хотите выполнять в фоновом режиме.
Log.e(" " + account.name, "Sync Start");
}
}

```

Служба синхронизации:

```

/**
 * Определите службу, которая возвращает IBinder для класса адаптера
 * синхронизации, позволяя фреймворку адаптера синхронизации вызывать
 * onPerformSync().
 */
public class SyncService extends Service {
    // Хранилище для экземпляра адаптера синхронизации
    private static SyncAdapter sSyncAdapter = null;
    // Объект для использования в качестве потокобезопасной блокировки
    private static final Object sSyncAdapterLock = new Object();

    /*
    Инстанцировать объект адаптера синхронизации.
    */
    @Override
    public void onCreate() {
        /*
        * Создайте адаптер синхронизации как синглтон.
        * Установите адаптер синхронизации как синхронизируемый
        * Запретите параллельную синхронизацию
        */
        synchronized (sSyncAdapterLock) {
            if (sSyncAdapter == null) {
                sSyncAdapter = new SyncAdapter(getApplicationContext(), true);
            }
        }
    }

    /**
     * Возвращает объект, позволяющий системе вызывать
     * адаптер синхронизации.
     */
    @Override
    public IBinder onBind(Intent intent) {
        /*
        * Получим объект, разрешающий внешние процессы
        * для вызова функции onPerformSync(). Объект создается
        * в коде базового класса, когда конструкторы SyncAdapter
        * вызывают super()
        */
    }
}

```

```
    return sSyncAdapter.getSyncAdapterBinder();
}
}
```

Аутентификатор:

```
public class Authenticator extends AbstractAccountAuthenticator {
    // Простой конструктор
    public Authenticator(Context context) {
        super(context);
    }

    // Редактирование свойств не поддерживается
    @Override
    public Bundle editProperties(
        AccountAuthenticatorResponse r, String s) {
        throw new UnsupportedOperationException();
    }

    // Не добавлять дополнительные аккаунты
    @Override
    public Bundle addAccount(
        AccountAuthenticatorResponse r,
        String s,
        String s2,
        String[] strings,
        Bundle bundle) throws NetworkErrorException {
        return null;
    }

    //忽орировать попытки подтверждения учетных данных
    @Override
    public Bundle confirmCredentials(
        AccountAuthenticatorResponse r,
        Account account,
        Bundle bundle) throws NetworkErrorException {
        return null;
    }

    // Получение токена аутентификации не поддерживается
    @Override
    public Bundle getAuthToken(
        AccountAuthenticatorResponse r,
        Account account,
        String s,
        Bundle bundle) throws NetworkErrorException {
        throw new UnsupportedOperationException();
    }

    // Получение метки для токена аутентификации не поддерживается
    @Override
    public String getAuthTokenLabel(String s) {
        throw new UnsupportedOperationException();
    }

    // Обновление учетных данных пользователя не поддерживается
    @Override
    public Bundle updateCredentials(
        AccountAuthenticatorResponse r,
```

```

        Account account,
        String s, Bundle bundle) throws NetworkErrorException {
    throw new UnsupportedOperationException();
}

// Функции проверки для данного счета не поддерживаются
@Override
public Bundle hasFeatures(
    AccountAuthenticatorResponse r,
    Account account, String[] strings) throws NetworkErrorException {
    throw new UnsupportedOperationException();
}
}

```

Служба аутентификатора:

```

/**
 * Связанная служба, которая инстанцирует аутентификатор при запуске.
 */
public class AuthenticatorService extends Service {
    // Поле экземпляра, в котором хранится объект аутентификатора
    private Authenticator mAuthenticator;
    @Override
    public void onCreate() {
        // Создание нового объекта аутентификатора
        mAuthenticator = new Authenticator(this);
    }
    /*
     * Когда система связывается с этой службой для выполнения RPC-вызова,
     * возвращается IBinder аутентификатора.
     */
    @Override
    public IBinder onBind(Intent intent) {
        return mAuthenticator.getIBinder();
    }
}

```

Дополнения к AndroidManifest.xml:

```

<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.READ_SYNC_SETTINGS" />
<uses-permission android:name="android.permission.WRITE_SYNC_SETTINGS" />
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />

<service
    android:name=".syncAdapter.SyncService"
    android:exported="true">
    <intent-filter>
        <action android:name="android.content.SyncAdapter" />
    </intent-filter>
    <meta-data
        android:name="android.content.SyncAdapter"
        android:resource="@xml/syncadapter" />
</service>

<service android:name=".authenticator.AuthenticatorService">
    <intent-filter>
        <action android:name="android.accounts.AccountAuthenticator" />
    </intent-filter>

```

```

<meta-data
    android:name="android.accounts.AccountAuthenticator"
    android:resource="@xml/authenticator" />
</service>

<provider
    android:name=".provider.StubProvider"
    android:authorities="com.yourpackage.provider"
    android:exported="false" android:syncable="true" />

```

res/xml/authenticator.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<account-authenticator xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="com.yourpackage"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:smallIcon="@mipmap/ic_launcher" />

```

res/xml/syncadapter.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<sync-adapter xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="com.yourpackage.android"
    android:allowParallelSyncs="false"
    android:contentAuthority="com.yourpackage.provider"
    android:isAlwaysSyncable="true"
    android:supportsUploading="false"
    android:userVisible="false" />

```

Провайдер заглушки StubProvider:

```

/*
 * Определите реализацию ContentProvider, которая будет заглушать
 * все методы
 */

public class StubProvider extends ContentProvider {
    /*
     * Всегда возвращает true, указывая на то,
     * что провайдер загрузился корректно.
     */
    @Override
    public boolean onCreate() {
        return true;
    }

    /*
     * Возврат отсутствия типа для типа MIME
     */
    @Override
    public String getType(Uri uri) {
        return null;
    }

    /*
     * query() всегда возвращает отсутствие результатов
     */
    @Override

```

```

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

/*
 * insert() всегда возвращает null (отсутствие URI)
 */
@Override
public Uri insert(Uri uri, ContentValues values) {
    return null;
}

/*
 * delete() всегда возвращает "строки не затронуты" (0)
 */
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    return 0;
}

/*
update() всегда возвращает "строки не затронуты" (0)
*/
public int update(
    Uri uri,
    ContentValues values,
    String selection,
    String[] selectionArgs) {
    return 0;
}
}

```

Вызов этой функции при успешном входе в систему для создания учетной записи с идентификатором:

```

public Account CreateSyncAccount(Context context, String accountName) {
    // Создание типа учетной записи и учетной записи по умолчанию
    Account newAccount = new Account( accountName, "com.yourpackage");
    // Получение экземпляра менеджера учетных записей Android
    AccountManager accountManager = (AccountManager) context.getSystemService(
        ACCOUNT_SERVICE);
    /*
     * Добавить учетную запись и ее тип, без пароля и данных о пользователе
     * В случае успеха возвращается объект Account, в противном случае сообщается
     * об ошибке.
     */
    if (accountManager.addAccountExplicitly(newAccount, null, null)) {
        /*
         * Если вы не устанавливаете android:syncable="true"
         * в элементе <provider> в манифесте,
         * то вызовите context.setIsSyncable(account, AUTHORITY, 1)
         * здесь.
         */
    } else {

```

```

    /*
     * Учетная запись существует или произошла какая-то другая ошибка.
     * Зафиксируйте это, сообщите об этом
     * или обработайте это внутренними средствами.
     */
}

return newAccount;
}

```

Принудительная синхронизация:

```

Bundle bundle = new Bundle();
bundle.putBoolean(ContentResolver.SYNC_EXTRAS_EXPEDITED, true);
bundle.putBoolean(ContentResolver.SYNC_EXTRAS_FORCE, true);
bundle.putBoolean(ContentResolver.SYNC_EXTRAS_MANUAL, true);
ContentResolver.requestSync(null, MyContentProvider.getAuthority(), bundle);

```

Глава 90. Режим Портера-Даффа

Это способ комбинирования изображений, как если бы они были “кусками картона неправильной формы”, наложенными друг на друга, а также схема смешивания наложенных друг на друга частей.

90.1. Создание цветового фильтра PorterDuff

Для создания фильтра `PorterDuffColorFilter` используется режим `PorterDuff.Mode` (см. <http://developer.android.com/reference/android/graphics/PorterDuffColorFilter.html>). Этот цветовой фильтр меняет цвет каждого пикселя визуального ресурса:

```
ColorFilter filter = new PorterDuffColorFilter(Color.BLUE, PorterDuff.Mode.SRC_IN);
```

Приведенный выше фильтр окрасит непрозрачные пиксели в синий цвет.
Цветовой фильтр может быть применен к `drawable`-объекту:

```
drawable.setColorFilter(filter);
```

Он может быть применен к `ImageView`:

```
imageView.setColorFilter(filter);
```

Кроме того, его можно применить к `Paint`, так что цвет, отрисованный этой краской, будет изменен фильтром:

```
paint.setColorFilter(filter);
```

90.2. Создание режима PorterDuff XferMode

Режим `Xfermode` (“перенос”) работает как шаг переноса в конвейере отрисовки. Когда режим `Xfermode` применяется к `Paint`, отрисовываемые пиксели комбинируются с нижележащими пикселями (уже отрисованными) в соответствии с режимом:

```
paint.setColor(Color.BLUE);
```

```
paint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.SRC_IN));
```

Теперь любая отрисованная фигура будет окрашивать внутри себя уже существующие непрозрачные пиксели в синий цвет.

90.3. Применение виньетки к растровому изображению с использованием PorterDuffXfermode

```
/*
 * Применим к растровому изображению радиальную маску (виньетку, то есть переход
 * к черному цвету по границам)
 * @param imageToApplyMaskTo Растровое изображение для модификации
 */
public static void radialMask(final Bitmap imageToApplyMaskTo) {
    Canvas canvas = new Canvas(imageToApplyMaskTo);

    final float centerX = imageToApplyMaskTo.getWidth() * 0.5f;
    final float centerY = imageToApplyMaskTo.getHeight() * 0.5f;
    final float radius = imageToApplyMaskTo.getHeight() * 0.7f;

    RadialGradient gradient = new RadialGradient(centerX, centerY, radius, 0x000000,
        0xFF000000, android.graphics.Shader.TileMode.CLAMP);
    Paint p = new Paint();
    p.setShader(gradient);
    p.setColor(0xFF000000);
    p.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.DST_OUT));
    canvas.drawRect(0, 0, imageToApplyMaskTo.getWidth(), imageToApplyMaskTo.
        getHeight(), p);
}
```

Глава 91. Меню

Параметр	Описание
inflate(int menuRes, Menu menu)	«Раздувает» иерархию меню из указанного XML-ресурса
getMenuInflater()	Возвращает MenuInflater с данным контекстом
onCreateOptionsMenu(Menu menu)	Инициализирует содержимое стандартного меню опций активности. Здесь указываются пункты меню
onOptionsItemSelected(MenuItem item)	Этот метод вызывается каждый раз, когда в меню опций выбирается пункт

91.1. Меню опций с разделителями

В Android по умолчанию имеется меню опций. Если необходимо отобразить большое количество опций, то для сохранения наглядности имеет смысл сгруппировать их. Это можно сделать, расположив между опциями разделители (то есть горизонтальные линии). Для того чтобы сделать разделители, можно использовать следующую тему:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Настроить тему можно здесь. -->
    <item name="colorPrimary">@color/colorPrimary</item>
```

```

<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
<item name="android:dropDownListStyle">@style/PopupMenuListView</item>
</style>
<style name="PopupMenuListView" parent="@style/Widget.AppCompat.ListView.DropDown">
    <item name="android:divider">@color/black</item>
    <item name="android:dividerHeight">1dp</item>
</style>

```

91.2. Применение пользовательского шрифта к меню

```

public static void applyFontToMenu(Menu m, Context mContext) {
    for(int i=0;i<m.size();i++) {
        applyFontToMenuItem(m.getItem(i),(mContext));
    }
}
public static void applyFontToMenuItem(MenuItem mi, Context mContext) {
    if(mi.hasSubMenu()) {
        for(int i=0;i<mi.getSubMenu().size();i++) {
            applyFontToMenuItem(mi.getSubMenu().getItem(i), mContext);
        }
    }
    Typeface font = Typeface.createFromAsset(mContext.getAssets(), "fonts/
yourCustomFont.ttf");
    SpannableString mNewTitle = new SpannableString(mi.getTitle());
    mNewTitle.setSpan(new CustomTypefaceSpan("", font, mContext), 0, mNewTitle.
length(), Spannable.SPAN_INCLUSIVE_INCLUSIVE);
    mi.setTitle(mNewTitle);
}

```

...а затем в активности:

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    applyFontToMenu(menu, this);
    return true;
}

```

91.3. Создание меню в активности

Чтобы определить собственное меню, создайте XML-файл в каталоге `res/menu/` вашего проекта и постройте меню из следующих элементов:

- `<menu>`: определяет меню, в котором содержатся все пункты меню.
- `<item>`: создает элемент `MenuItem`, который представляет собой один пункт меню. Можно также создать вложенный элемент для формирования подменю.

Шаг 1

Создайте свой собственный xml-файл следующим образом.

В файле `res/menu/main_menu.xml` используйте код:

```

<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/aboutMenu"
        android:title="About" />

```

```

<item
    android:id="@+id/helpMenu" android:title="Help" />
<item
    android:id="@+id/signOutMenu" android:title="Sign Out" />
</menu>

```

Шаг 2

Чтобы настроить меню опций, переопределите в своей активности функцию `onCreateOptionsMenu()`. В этом методе вы можете «раздуть» ресурс меню (определенный в вашем XML-файле, то есть `res/menu/main_menu.xml`):

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_menu, menu);
    return true;
}

```

Когда пользователь выбирает пункт из меню опций, система вызывает переопределенный метод `onOptionsItemSelected()` вашей активности.

- Этот метод передает выбранный `MenuItem`.
- Идентифицировать пункт можно, вызвав функцию `getItemId()`, которая возвращает уникальный идентификатор пункта меню (определяется атрибутом `android:id` в ресурсе меню: `res/menu/main_menu.xml`)*.

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.aboutMenu:
            Log.d(TAG, "Нажал на About!");
            // Код для About находится здесь
            return true;
        case R.id.helpMenu:
            Log.d(TAG, "Нажал на Help!");
            // Код для Help находится здесь
            return true;
        case R.id.signOutMenu:
            Log.d(TAG, "Нажал на Sign Out!");
            // Вызов метода SignOut находится здесь
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

Завершающая часть

Ваш код для `Activity` должен выглядеть следующим образом:

```

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "mytag";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

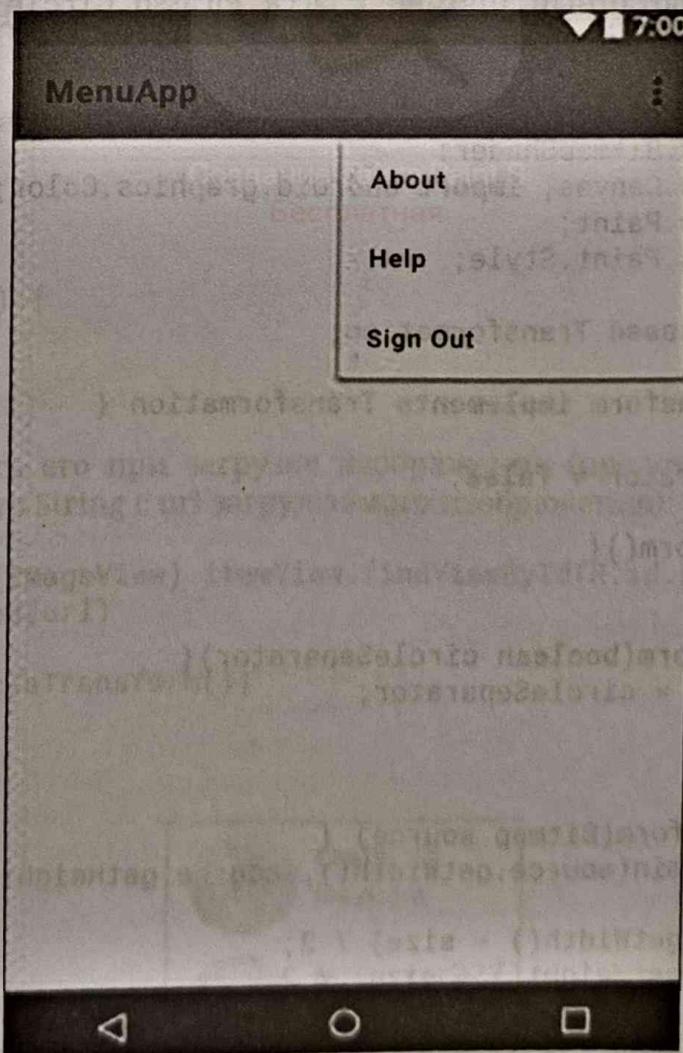
```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.aboutMenu:
            Log.d(TAG, "Нажал на About!");
            // Код для About находится здесь
            return true;
        case R.id.helpMenu:
            Log.d(TAG, "Нажал на Help!");
            // Код для справки находится здесь
            return true;
        case R.id.signOutMenu:
            Log.d(TAG, "Пользователь вышел из системы");
            // Вызов метода SignOut находится здесь
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
}

```

Скриншот меню:



Глава 92. Picasso

Picasso (<http://square.github.io/picasso/>) – это библиотека изображений для Android, созданная и поддерживаемая компанией Square (<http://square.github.io/>). Она упрощает процесс отображения изображений из внешних источников. Библиотека обрабатывает все этапы процесса, начиная с первоначального HTTP-запроса и заканчивая кэшированием изображения. Во многих случаях для использования этой удобной библиотеки требуется всего несколько строк кода.

92.1. Добавление библиотеки Picasso в проект

Gradle:

```
dependencies {
    compile "com.squareup.picasso:picasso:2.5.2"
}
```

Maven:

```
<dependency>
    <groupId>com.squareup.picasso</groupId>
    <artifactId>picasso</artifactId>
    <version>2.5.2</version>
</dependency>
```

92.2. Создание круглых аватаров с помощью Picasso

Здесь приведен доработанный пример класса Picasso CircleTransform, основанного на примере с ресурса <https://gist.github.com/julianshen/5829333>:

```
import android.graphics.Bitmap;
import android.graphics.BitmapShader;
import android.graphics.Canvas; import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Paint.Style;

import com.squareup.picasso.Transformation;

public class CircleTransform implements Transformation {

    boolean mCircleSeparator = false;

    public CircleTransform(){
    }

    public CircleTransform(boolean circleSeparator){
        mCircleSeparator = circleSeparator;
    }

    @Override
    public Bitmap transform(Bitmap source) {
        int size = Math.min(source.getWidth(), source.getHeight());

        int x = (source.getWidth() - size) / 2;
        int y = (source.getHeight() - size) / 2;

        Bitmap squaredBitmap = Bitmap.createBitmap(source, x, y, size, size);
```

```

if (squaredBitmap != source) {
    source.recycle();
}

Bitmap bitmap = Bitmap.createBitmap(size, size, source.getConfig());

Canvas canvas = new Canvas(bitmap);
BitmapShader shader = new BitmapShader(squaredBitmap, BitmapShader.TileMode.
CLAMP, BitmapShader.TileMode.CLAMP);
Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG | Paint.DITHER_FLAG | Paint.
FILTER_BITMAP_FLAG);
paint.setShader(shader);

float r = size/2f; canvas.drawCircle(r, r, r-1, paint);

// Сделать тонкую границу:
Paint paintBorder = new Paint();
paintBorder.setStyle(Style.STROKE);
paintBorder.setColor(Color.argb(84, 0, 0, 0));
paintBorder.setAntiAlias(true);
paintBorder.setStrokeWidth(1);
canvas.drawCircle(r, r, r-1, paintBorder);

// Необязательный разделитель для укладывания в стопку:
if (mCircleSeparator) {
    Paint paintBorderSeparator = new Paint();
    paintBorderSeparator.setStyle(Style.STROKE);
    paintBorderSeparator.setColor(Color.parseColor("#ffffff"));
    paintBorderSeparator.setAntiAlias(true);
    paintBorderSeparator.setStrokeWidth(4);
    canvas.drawCircle(r, r, r+1, paintBorderSeparator);
}

squaredBitmap.recycle();
return bitmap;
}

@Override
public String key() {
    return "круг";
}
}

```

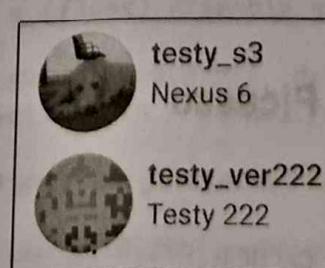
Вот как использовать его при загрузке изображения (предполагается, что `this` – это Activity Context, а `url` – это String с url загружаемого изображения):

```

ImageView ivAvatar = (ImageView) itemView.findViewById(R.id.avatar);
Picasso.with(this).load(url)
    .fit()
    .transform(new CircleTransform())
    .into(ivAvatar);

```

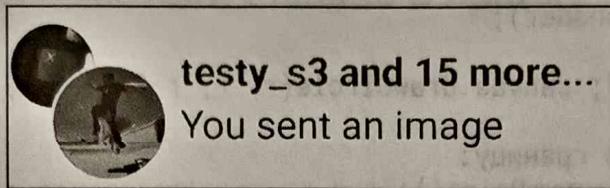
Результат:



Для использования с разделителем укажите `true` в конструкторе для верхнего изображения:

```
ImageView ivAvatar = (ImageView) itemView.findViewById(R.id.avatar);
Picasso.with(this).load(url)
    .fit()
    .transform(new CircleTransform(true))
    .into(ivAvatar);
```

Результат (два ImageView в FrameLayout):



92.3. Местозаполнители

Picasso поддерживает опциональные возможности местозаполнителей при загрузке и для ошибок. Кроме того, он предоставляет обратные вызовы для обработки результатов загрузки.

```
Picasso.with(context)
    .load("URL ВАШЕГО ИЗОБРАЖЕНИЯ ЗДЕСЬ")
    .placeholder(Your Drawable Resource) //это необязательное изображение, которое
    должно отображаться в качестве url-изображения, пока оно загружается
    .error(Your Drawable Resource) // это также необязательно – если произошла
    какая-то ошибка при загрузке изображения, на экран будет выведено следующее
    изображение
    .into(imageView, new Callback(){
        @Override
        public void onSuccess() {}

        @Override
        public void onError() {}
    });
}
```

Запрос будет повторен три раза, прежде чем будет показан этот местозаполнитель при ошибке.

92.4. Изменение размеров и поворот

```
Picasso.with(context)
    .load("URL ВАШЕГО ИЗОБРАЖЕНИЯ ЗДЕСЬ")
    .placeholder(DRAWABLE RESOURCE)           // опционально
    .error(DRAWABLE RESOURCE)                 // опционально
    .resize(width, height)                   // опционально
    .rotate(degree)                         // опционально
    .into(imageView);
```

92.5. Отключение кэша в Picasso

```
Picasso.with(context)
    .load(uri)
    .networkPolicy(NetworkPolicy.NO_CACHE)
```

```
.memoryPolicy(MemoryPolicy.NO_CACHE)
.placeholder(R.drawable.placeholder)
.into(imageView);
```

92.6. Использование Picasso в качестве ImageGetter для Html.fromHtml

```
public class PicassoImageGetter implements Html.ImageGetter {
    private TextView textView;
    private Picasso picasso;

    public PicassoImageGetter(@NonNull Picasso picasso, @NonNull TextView textView) {
        this.picasso = picasso;
        this.textView = textView;
    }

    @Override
    public Drawable getDrawable(String source) {
        Log.d(PicassoImageGetter.class.getName(), "Начало загрузки url " + source);
        BitmapDrawablePlaceHolder drawable = new BitmapDrawablePlaceHolder();
        picasso
            .load(source)
            .error(R.drawable.connection_error)
            .into(drawable);

        return drawable;
    }

    private class BitmapDrawablePlaceHolder extends BitmapDrawable implements Target {
        protected Drawable drawable;

        @Override
        public void draw(Canvas canvas) {
            if (drawable != null) {
                checkBounds();
                drawable.draw(canvas);
            }
        }

        public void setDrawable(@Nullable Drawable drawable) {
            if (drawable != null) {
                this.drawable = drawable;
                checkBounds();
            }
        }

        private void checkBounds() {
            float defaultProportion = (float) drawable.getIntrinsicWidth() / (float)
            drawable.getIntrinsicHeight();
            int width = Math.min(textView.getWidth(), drawable.getIntrinsicWidth());
            int height = (int) ((float) width / defaultProportion);

            if (getBounds().right != textView.getWidth() || getBounds().bottom != height) {
                setBounds(0, 0, textView.getWidth(), height); //установка на полную ширину
            }
        }
    }
}
```

```

        int halfOfPlaceHolderWidth = (int) ((float) getBounds().right / 2f);
        int halfOfImageWidth = (int) ((float) width / 2f);

        drawable.setBounds(
            halfOfPlaceHolderWidth - halfOfImageWidth, //центрирование изображения
            0,
            halfOfPlaceHolderWidth + halfOfImageWidth, height);

        textView.setText(textView.getText()); //обновить текст
    }
}

// -----
@Override
public void onBitmapLoaded(Bitmap bitmap, Picasso.LoadedFrom from) {
    setDrawable(new BitmapDrawable(Application.getContext().getResources(), bitmap));
}

@Override
public void onBitmapFailed(Drawable errorDrawable) {
    setDrawable(errorDrawable);
}

@Override
public void onPrepareLoad(Drawable placeHolderDrawable) {
    setDrawable(placeHolderDrawable);
}

// -----
}
}

```

Использование:

```
Html.fromHtml(textToParse, new PicassoImageGetter(picasso, textViewTarget), null);
```

92.7. Отмена запросов изображений с помощью Picasso

В некоторых случаях нам необходимо отменить запрос на загрузку изображения в Picasso до завершения загрузки.

Это может произойти по разным причинам, например, если родительский вид перешел к другому виду до завершения загрузки изображения.

В этом случае запрос на загрузку изображения можно отменить с помощью метода `cancelRequest()`:

```

ImageView imageView;
//.....
Picasso.with(imageView.getContext()).cancelRequest(imageView);

```

92.8. Загрузка изображения из внешнего хранилища

```

String filename = "image.png";
String imagePath = getExternalFilesDir() + "/" + filename;

```

```
Picasso.with(context)
    .load(new File(imagePath))
    .into(imageView);
```

92.9. Загрузка изображения как Bitmap с помощью Picasso

Если вы хотите загрузить изображение как Bitmap с помощью Picasso, то вам поможет следующий код:

```
Picasso.with(mContext)
    .load(ImageUrl)
    .into(new Target() {
        @Override
        public void onBitmapLoaded(Bitmap bitmap, Picasso.LoadedFrom from) {
            // Todo: Сделайте что-нибудь с вашим растровым изображением здесь
        }

        @Override
        public void onBitmapFailed(Drawable errorDrawable) {
        }

        @Override
        public void onPrepareLoad(Drawable placeHolderDrawable) {
        }
    });
}
```

Глава 93. RoboGuice

93.1. Простой пример

RoboGuice – это фреймворк, который обеспечивает простоту и легкость внедрения зависимостей (Dependency Injection) в Android, используя библиотеку Guice от Google:

```
@ContentView(R.layout.main)
class RoboWay extends RoboActivity {
    @InjectView(R.id.name)           TextView name;
    @InjectView(R.id.thumbnail)       ImageView thumbnail;
    @InjectResource(R.drawable.icon) Drawable icon;
    @InjectResource(R.string.app_name) String myName;
    @Inject                           LocationManager loc;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        name.setText("Здравствуйте, " + myName );
    }
}
```

93.2. Установка для проектов Gradle

Добавьте это в раздел зависимостей вашего файла сборки gradle:

```
project.dependencies {
    compile 'org.robolectric:robolectric:3.+'
    provided 'org.robolectric:robolectric:3.+'
}
```

93.3. Аннотация @ContentView

Аннотация `@ContentView` может быть использована для дальнейшего упрощения разработки активностей и замены оператора `setContentView`:

```
@ContentView(R.layout.myactivity_layout)
public class MyActivity extends RoboActivity {
    @InjectView(R.id.text1) TextView textView;

    @Override
    protected void onCreate( Bundle savedInstanceState ) {
        textView.setText("Hello!");
    }
}
```

93.4. Аннотация @InjectResource

Вы можете внедрять зависимости любого типа: строки, анимации, Drawable-объекты и т. д. Чтобы внедрить свой первый ресурс в активность, необходимо:

- Наследовать от `RoboActivity`.
- Аннотировать свои ресурсы с помощью `@InjectResource`.

Пример:

```
@InjectResource(R.string.app_name) String name;
@InjectResource(R.drawable.ic_launcher) Drawable icLauncher;
@InjectResource(R.anim.my_animation) Animation myAnimation;
```

93.5. Аннотация @InjectView

Любое представление можно внедрить с помощью аннотации `@InjectView`.

Для этого необходимо:

- Наследовать от `RoboActivity`.
- Настроить представление своего содержимого.
- Аннотировать свои представления с помощью `@InjectView`.

Пример:

```
@InjectView(R.id.textView1) TextView textView1;
@InjectView(R.id.textView2) TextView textView2;
@InjectView(R.id.imageView1) ImageView imageView1;
```

93.6. Введение в RoboGuice

RoboGuice 3 позволяет сократить объем кода приложения. Меньше кода – меньше возможностей для ошибок. Кроме того, код становится более понятным – он больше не засоряется механикой платформы Android, а концентрируется на логике, присущей только вашему приложению.

Чтобы получить представление о полезности RoboGuice, рассмотрим простой пример типичной активности в Android:

```
class AndroidWay extends Activity {
```

```

    TextView name;
    ImageView thumbnail;
    LocationManager loc;
    Drawable icon;
    String myName;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        name      = (TextView) findViewById(R.id.name);
        thumbnail = (ImageView) findViewById(R.id.thumbnail);
        loc       = (LocationManager) getSystemService(Activity.LOCATION_SERVICE);
        icon      = getResources().getDrawable(R.drawable.icon);
        myName   = getString(R.string.app_name);
        name.setText("Здравствуйте, " + myName );
    }
}

```

Этот пример состоит из 19 строк кода. Если вы попытаетесь прочитать `onCreate()`, то вам придется пропустить 5 строк шаблонной инициализации, чтобы найти единственную, которая действительно имеет значение: `name.setText()`. А сложные активности могут содержать гораздо больше подобного инициализационного кода.

Сравните это с тем же приложением, написанным с помощью RoboGuice:

```

@ContentView(R.layout.main)
class RoboWay extends RoboActivity {
    @InjectView(R.id.name)           TextView name;
    @InjectView(R.id.thumbnail)       ImageView thumbnail;
    @InjectResource(R.drawable.icon) Drawable icon;
    @InjectResource(R.string.app_name) String myName;
    @Inject                         LocationManager loc;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        name.setText("Здравствуйте, " + myName );
    }
}

```

Цель RoboGuice – сделать так, чтобы ваш код был ориентирован на ваше приложение, а не на весь код инициализации и жизненного цикла, который обычно приходится поддерживать в Android.

Аннотации

Аннотация @ContentView

Может быть использована для дальнейшего упрощения разработки активностей и замены оператора `setContentView`:

```

@ContentView(R.layout.myactivity_layout)
public class MyActivity extends RoboActivity {
    @InjectView(R.id.text1) TextView textView;

    @Override
    protected void onCreate( Bundle savedState ) {
        textView.setText("Hello!");
    }
}

```

Аннотация @InjectResource

Сначала необходимо создать активность, наследующую от RoboActivity. Затем, предположив, что в папке res/anim имеется анимация my_animation.xml, можно сослаться на нее с помощью аннотации:

```
public class MyActivity extends RoboActivity {
    @InjectResource(R.anim.my_animation) Animation myAnimation;
    // остальной код
}
```

Помимо представлений, ресурсов, служб и других специфических для Android вещей RoboGuice может внедрять POJO-объекты (Plain Old Java Objects, «обычные старые Java-объекты»). По умолчанию Roboguice будет вызывать для вашего POJO конструктор без аргументов:

```
class MyActivity extends RoboActivity {
    @Inject Foo foo; // это вызовет новое Foo();
}
```

Глава 94. Библиотека ACRA

Параметр Описание

@ReportCrashes	Определяет настройки ACRA, такие как место, куда идут сообщения, пользовательское содержимое и т. д.
formUri	Путь к файлу, в котором сообщается об аварийном завершении

94.1. ACRAHandler

Пример класса расширения приложения для работы с отчетностью:

```
@ReportsCrashes(
    formUri = "https://backend-of-your-choice.com/", //Защита от непаролей.
    customReportContent = { /* */ReportField.APP_VERSION_NAME, ReportField.
        PACKAGE_NAME, ReportField.ANDROID_VERSION, ReportField.PHONE_
        MODEL, ReportField.LOGCAT },
    mode = ReportingInteractionMode.TOAST,
    restToastText = R.string.crash
)
public class ACRAHandler extends Application {
    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);

        final ACRAConfiguration config = new ConfigurationBuilder(this)
            .build();

        // Инициализация ACRA
        ACRA.init(this, config);
    }
}
```

94.2. Пример манифеста

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    <!-- И т.д. -->
    > private String owner;
    > private boolean isPrivate;

    <!-- Требуется наличие Интернета. READ_LOGS предназначены для обеспечения
    передачи Logcat-->
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_LOGS"/>

    <application
        android:allowBackup="true"
        android:name=".ACRAHandler"<!-- Активирует ACRA при запуске -->
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <!-- Активности -->
    </application>
</manifest>

```

94.3. Установка

Maven:

```

<dependency>
    <groupId>ch.acra</groupId>
    <artifactId>acra</artifactId>
    <version>4.9.2</version>
    <type>aar</type>
</dependency>

```

Gradle:

```
compile 'ch.acra:acra:4.9.2'
```

Глава 95. Parcelable

Parcelable – это специфический интерфейс, в котором вы сами реализуете сериализацию. Он был создан для того, чтобы получить большую эффективность, чем Serializable, и обойти некоторые проблемы стандартной схемы сериализации Java.

95.1. Создание пользовательского Parcelable-объекта

```

/**
 * Создано Алексом Салливаном.
 */
public class Foo implements Parcelable
{
    private final int myFirstVariable;
    private final String mySecondVariable;
}

```

```
private final long myThirdVariable;

public Foo(int myFirstVariable, String mySecondVariable, long myThirdVariable) {
    this.myFirstVariable = myFirstVariable;
    this.mySecondVariable = mySecondVariable;
    this.myThirdVariable = myThirdVariable;
}

// Обратите внимание, что читать значения из посылки (parcel) нужно в том же
// порядке, в котором они были ЗАПИСАНЫ в посылку! Этот метод является нашим
// собственным пользовательским методом для инстанцирования нашего объекта
// из посылки. Он используется в переменной Parcelable.Creator, которую мы
// объявляем ниже.
public Foo(Parcel in)
{
    this.myFirstVariable = in.readInt();
    this.mySecondVariable = in.readString();
    this.myThirdVariable = in.readLong();
}

// Метод describeContents обычно может возвращать 0. Он используется, когда
// объект посылки включает дескриптор файла.
@Override
public int describeContents()
{
    return 0;
}

@Override
public void writeToParcel(Parcel dest, int flags)
{
    dest.writeInt(myFirstVariable);
    dest.writeString(mySecondVariable);
    dest.writeLong(myThirdVariable);
}

// Обратите внимание, что это кажущееся случайным поле НЕ ЯВЛЯЕТСЯ ВАРИАНТОМ.
// Система будет искать эту переменную с помощью рефлексии, чтобы инстанцировать
// свой парселированный объект при чтении из намерения.
public static final Parcelable.Creator<Foo> CREATOR = new Parcelable.
Creator<Foo>() {
    // Этот метод используется для фактического инстанцирования нашего
    // пользовательского объекта из посылки (Parcel). Конвенция предписывает нам
    // сделать новый конструктор, который принимает посылку в качестве
    // единственного аргумента.
    public Foo createFromParcel(Parcel in)
    {
        return new Foo(in);
    }

    // Этот метод используется для создания массива вашего пользовательского
    // объекта. Объявления нового массива с указанным размером обычно бывает
    // достаточно.
    public Foo[] newArray(int size)
    {
        return new Foo[size];
    }
};
```

95.2. Parcelable-объект, содержащий другой Parcelable-объект

Пример класса, содержащего внутри себя посыльный (Parcelable) класс:

```
public class Repository implements Parcelable {
    private String name;
    private Owner owner;
    private boolean isPrivate;

    public Repository(String name, Owner owner, boolean isPrivate) {
        this.name = name;
        this.owner = owner;
        this.isPrivate = isPrivate;
    }

    protected Repository(Parcel in) {
        name = in.readString();
        owner = in.readParcelable(Owner.class.getClassLoader());
        isPrivate = in.readByte() != 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(name);
        dest.writeParcelable(owner, flags);
        dest.writeByte((byte) (isPrivate ? 1 : 0));
    }

    @Override
    public int describeContents() {
        return 0;
    }

    public static final Creator<Repository> CREATOR = new Creator<Repository>() {
        @Override
        public Repository createFromParcel(Parcel in) {
            return new Repository(in);
        }

        @Override
        public Repository[] newArray(int size) {
            return new Repository[size];
        }
    };

    //Геттеры и сеттеры

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Owner getOwner() {
        return owner;
    }
}
```

```

public void setOwner(Owner owner) {
    this.owner = owner;
}

public boolean isPrivate() {
    return isPrivate;
}

public void setPrivate(boolean isPrivate) {
    this.isPrivate = isPrivate;
}
}

```

Owner – это обычный Parcelable-класс.

95.3. Использование Enums с Parcelable-объектами

```

/**
Создано Ником Кардосо.
Это не полная реализация parcelable, а только самые простые
способы чтения и записи значений Enum в вашу посылку (parcel).
*/
public class Foo implements Parcelable {

    private final MyEnum myEnumVariable;
    private final MyEnum mySaferEnumVariableExample;

    public Foo(Parcel in) {
        //простейший способ
        myEnumVariable = MyEnum.valueOf( in.readString() );

        //с некоторой проверкой ошибок
        try {
            mySaferEnumVariableExample= MyEnum.valueOf( in.readString() );
        } catch (IllegalArgumentException e) { //неправильная строка или нулевое
            значение
            mySaferEnumVariableExample= MyEnum.DEFAULT;
        }
    }

    ...
    @Override
    public void writeToParcel(Parcel dest, int flags) {
        //простой способ
        dest.writeString(myEnumVariable.name());

        //избежание NPE с помощью некоторой проверки ошибок
        dest.writeString(mySaferEnumVariableExample == null? null :
            mySaferEnumVariableExample.name());
    }
}

```

```
public enum MyEnum {
    VALUE_1,
    VALUE_2,
    DEFAULT
}
```

Это предпочтительнее, чем (например) использование порядкового номера, поскольку вставка новых значений в перечисление не повлияет на ранее сохраненные значения.

Глава 96. Пикеры даты и времени

96.1. Диалоговое окно выбора даты Date Picker

Это диалоговое окно, предлагающее пользователю выбрать дату с помощью DatePicker. Для диалога с начальной датой требуется контекст, начальные год, месяц и день. Когда пользователь выбирает дату, происходит обратный вызов через DatePickerDialog.OnDateSetListener.

```
public void showDatePicker(Context context, int initialYear, int initialMonth, int
initialDay) {
    DatePickerDialog datePickerDialog = new DatePickerDialog(context,
        new DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker datepicker, int year, int month, int day) {
                //это условие необходимо для корректной работы на всех версиях android
                if(view.isShown()){
                    //Теперь у вас есть выбранные год, месяц и день
                }
            }
        },
        initialYear, initialMonth, initialDay);

    //Вызовите show(), чтобы показать диалог
    datePickerDialog.show();
}
```

Глава 97. Покалипсанные даты

Обратите внимание, что месяц – это целое число int, начинающееся от 0 для января до 11 для декабря.

96.2. DatePicker с использованием материального оформления

Добавьте в файл build.gradle в секцию dependencies следующие зависимости (это неофициальная библиотека для выбора даты):

```
compile 'com.wdullaer:materialdatetimepicker:2.3.0'
```

Теперь нам нужно открыть DatePicker по событию клика по кнопке. Для этого создайте одну кнопку в xml-файле, как показано ниже.

```
<Button
    android:id="@+id/dialog_bt_date"
```

```
    android:layout_below="@+id/resetButton"
    android:layout_width="wrap_content"
    android:layout_height="40dp"
    android:textColor="#FF000000"
    android:gravity="center" android:text="DATE" />
```

А в MainActivity используйте следующий способ:

```
public class MainActivity extends AppCompatActivity implements DatePickerDialog.OnDateSetListener{

    Button button;
    Calendar calendar ;
    DatePickerDialog datePickerDialog ;
    int Year, Month, Day ;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        calendar = Calendar.getInstance();

        Year = calendar.get(Calendar.YEAR) ;
        Month = calendar.get(Calendar.MONTH) ;
        Day = calendar.get(Calendar.DAY_OF_MONTH) ;

        private void showDatePicker() {
            Button dialog_bt_date = (Button)findViewById(R.id.dialog_bt_date); dialog_bt_
            date.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {

                    datePickerDialog = DatePickerDialog.newInstance(MainActivity.this,
                        Year, Month, Day);

                    datePickerDialog.setThemeDark(false);
                    datePickerDialog.showYearPickerFirst(false);
                    datePickerDialog.setAccentColor(Color.parseColor("#0072BA"));
                    datePickerDialog.setTitle("Select Date From DatePickerDialog");
                    datePickerDialog.show(getFragmentManager(), "DatePickerDialog");
                }
            });
        }

        @Override
        public void onDateSet(DatePickerDialog view, int Year, int Month, int Day) {
            String date = "Selected Date : " + Day + "-" + Month + "-" + Year;
            Toast.makeText(MainActivity.this, date, Toast.LENGTH_LONG).show();
        }
        @Override
        public boolean onCreateOptionsMenu(Menu menu)
        {
            getMenuInflater().inflate(R.menu.abc_main_menu, menu);
            return true;
        }
    }
```