

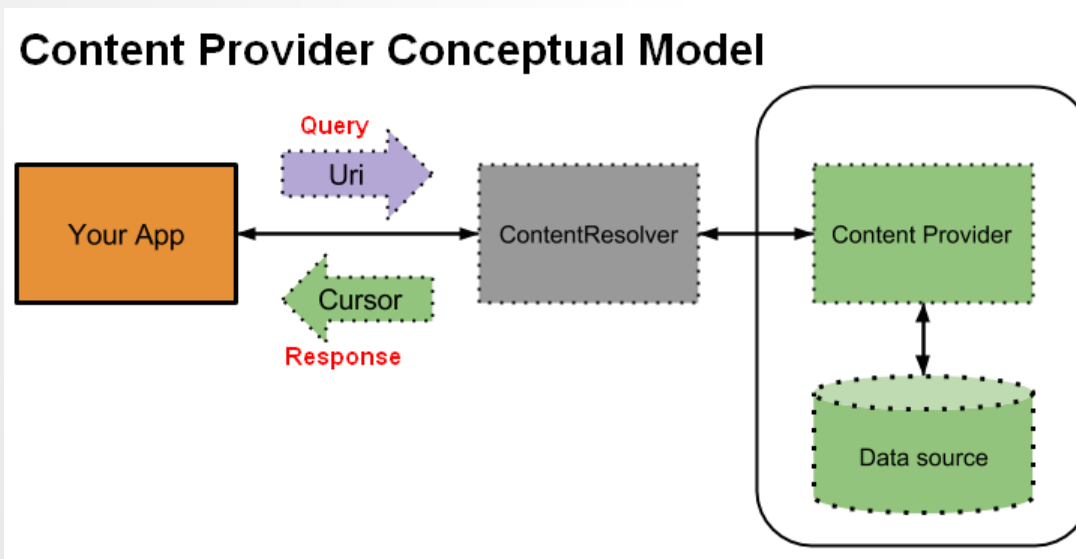
A faint, light gray world map is visible in the background, centered on the Atlantic Ocean. The map shows the outlines of continents and major islands.

# ANDROID移动互联网应用开发

傅晓，河海大学计算机与信息学院  
2019年3月

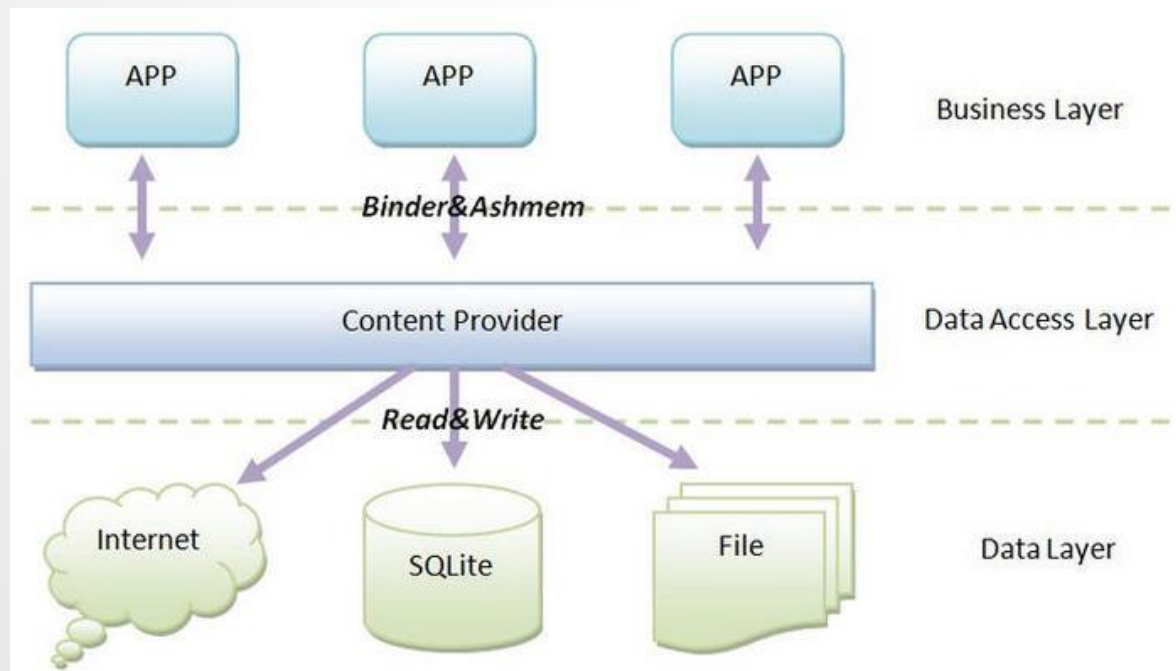
# CONTENT PROVIDER

- ContentProvider即内容提供者，是Android四大组件之一。
- 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内置的许多数据都是使用ContentProvider形式，供开发者调用的 (如视频，音频，图片，通讯录等)。



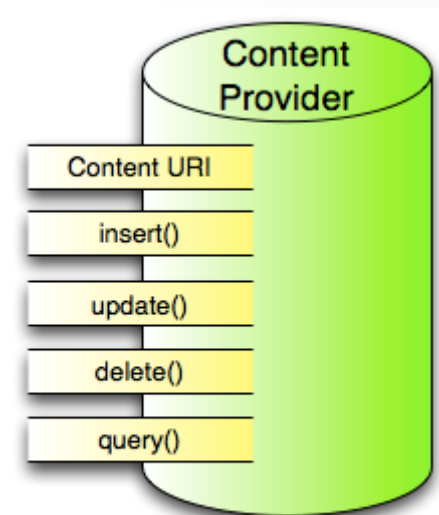
# CONTENT PROVIDER

- ContentProvider提供了对底层数据存储方式的抽象。
- Android框架中的一些类需要ContentProvider类型数据。
- ContentProvider为应用间的数据交互提供了一个安全的环境。



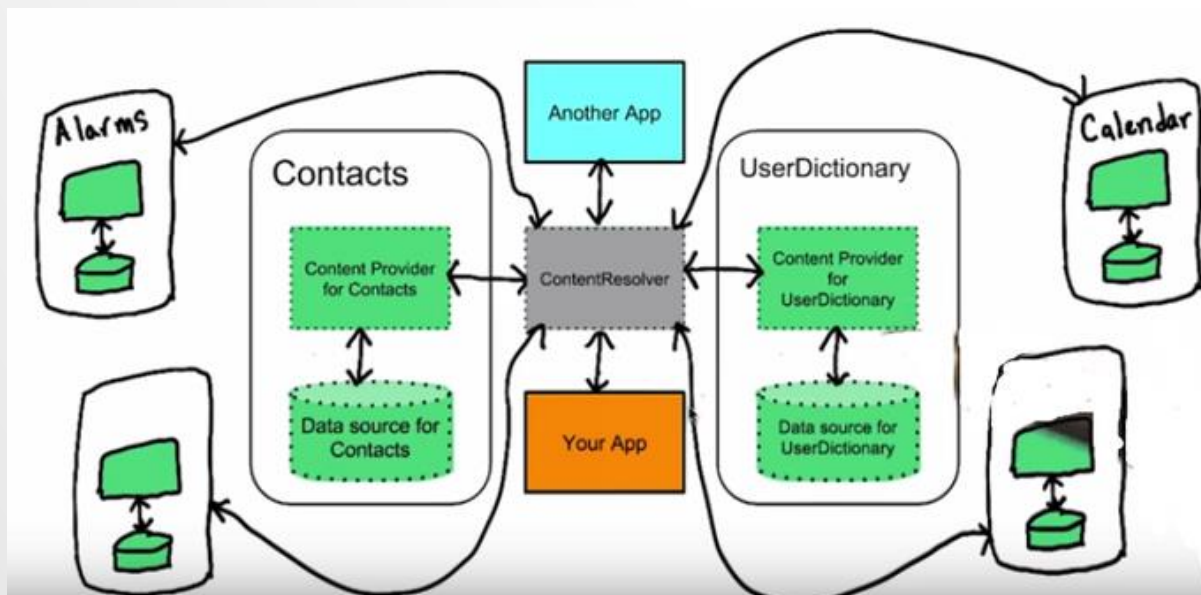
# CONTENT PROVIDER

- ContentProvider提供以下方法：
- query：查询；
- insert：插入；
- update：更新；
- delete：删除；
- getType：得到数据类型；
- onCreate：创建数据时调用的回调函数。



# CONTENT PROVIDER

- ContentResolver即内容解析器，程序通过ContentResolver可以访问ContentProvider提供的数据库。
- 使用ContentResolver对ContentProvider进行增、删、改、查的操作，应用程序开发者不需要知道ContentProvider的内部实现，ContentResolver可统一管理与不同ContentProvider间的操作。



# CONTENT PROVIDER

- 每个ContentProvider都有一个公共的URI，这个URI用于表示这个ContentProvider所提供的数据库。
- Android所提供的ContentProvider都存放在android.provider包当中。
- ContentResolver通过URI来区别不同的ContentProvider。

**自定义URI = content:// com.carson.provider / User / 1**

主题名

授权信息

表名

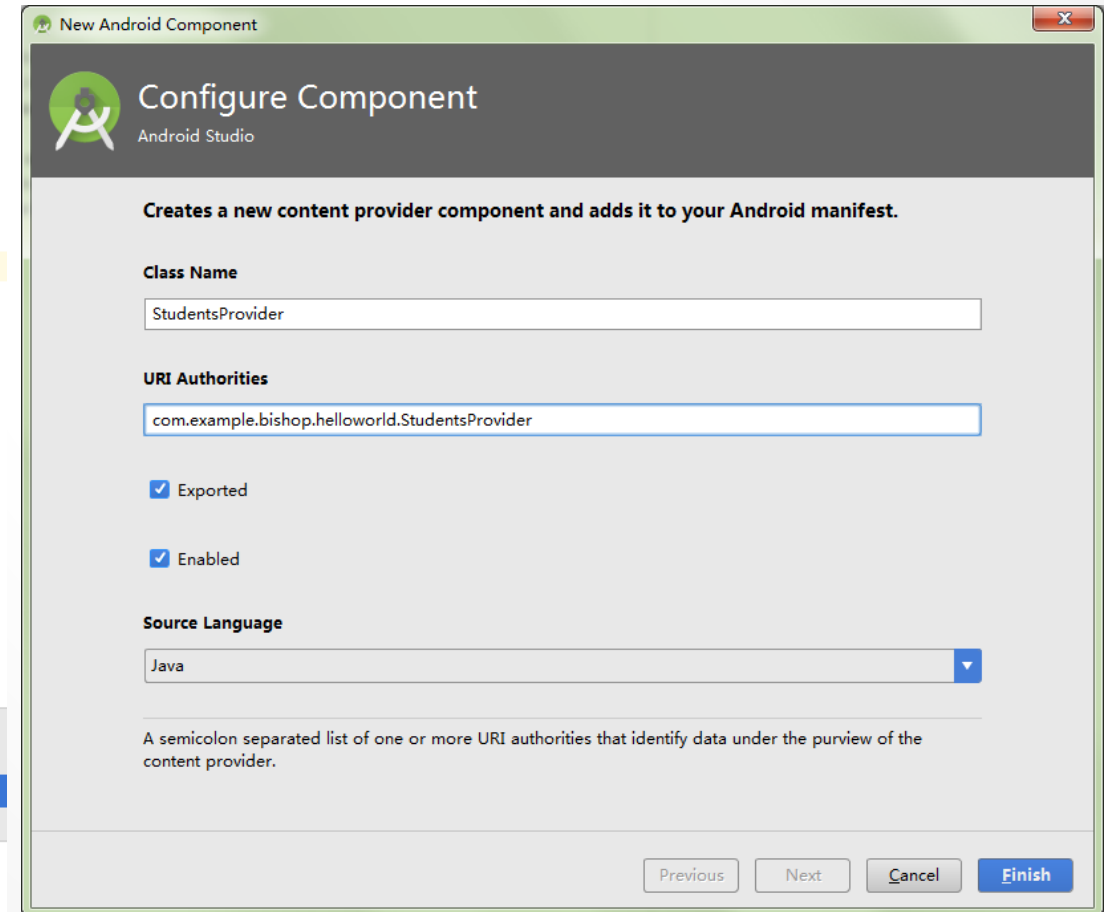
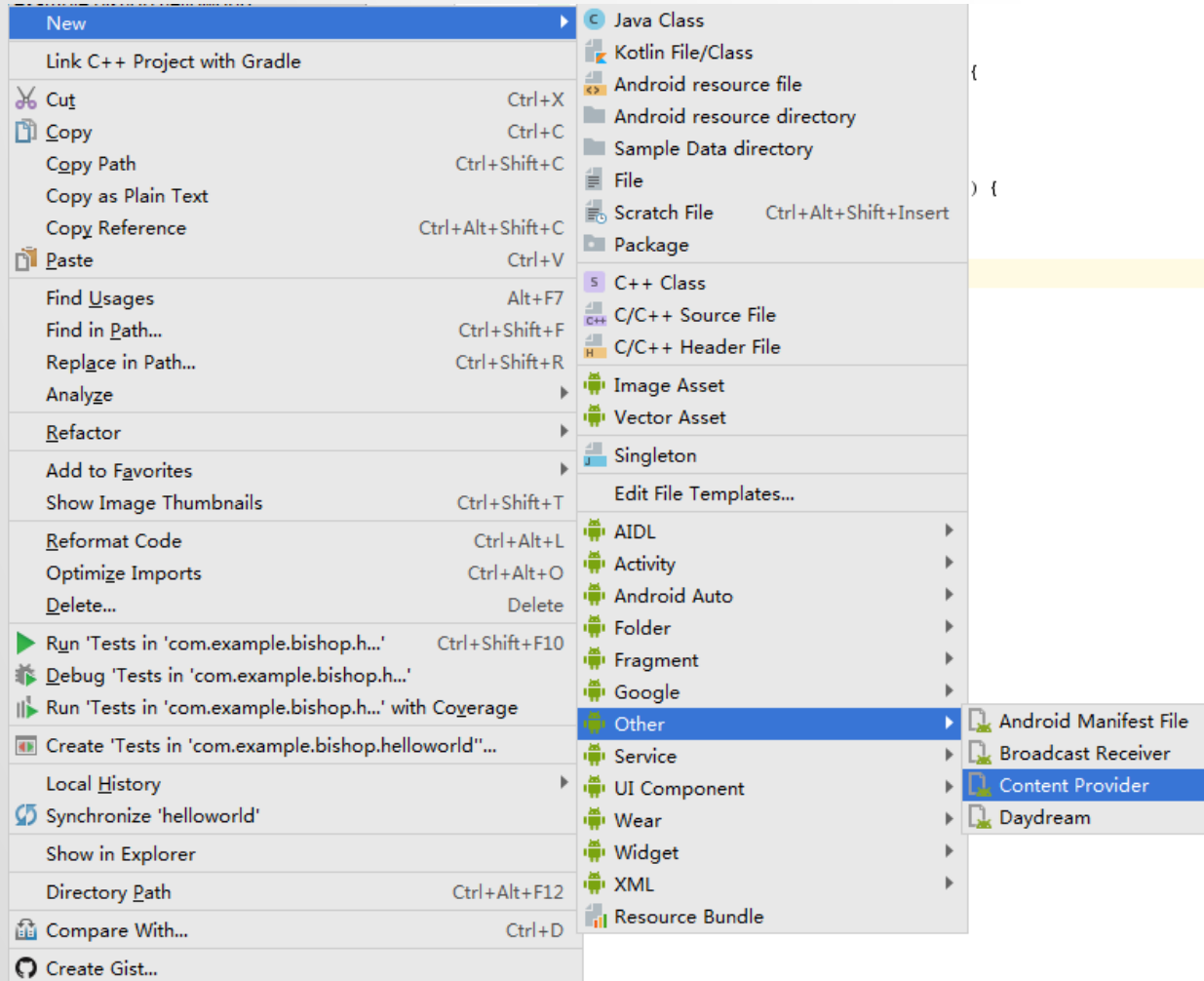
记录

- 主题 (Schema) : Content Provider的URI前缀 (Android 规定)
- 授权信息 (Authority) : Content Provider的唯一标识符
- 表名 (Path) : Content Provider 指向数据库中的某个表名
- 记录 (ID) : 表中的某个记录 (若无指定, 则返回全部记录)

# CONTENT PROVIDER

- URI (Universal Resource Identifier, 统一资源定位符) 代表要操作的数据, Android上可用的每种资源, 图像、视频片段等都可以用URI来表示:
- 所有联系人的URI : `content://contacts/people`
- 某个联系人的URI : `content://contacts/people/5`
- 所有图片URI : `content://media/external`
- 某个图片的URI : `content://media/external/images/media/4`

# CONTENT PROVIDER





# CONTENT PROVIDER

```
package com.example.bishop.helloworld;

import ...

public class StudentsProvider extends ContentProvider {
    public StudentsProvider() {
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        // Implement this to handle requests to delete one or more rows.
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public String getType(Uri uri) {
        // TODO: Implement this to handle requests for the MIME type of the data
        // at the given URI.
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        // TODO: Implement this to handle requests to insert a new row.
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public boolean onCreate() {
        // TODO: Implement this to initialize your content provider on startup.
        return false;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
        String[] selectionArgs, String sortOrder) {
        // TODO: Implement this to handle query requests from clients.
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
        String[] selectionArgs) {
        // TODO: Implement this to handle requests to update one or more rows.
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```

```
package com.example.bishop.helloworld;

import ...

public class StudentsProvider extends ContentProvider {
    static final String PROVIDER_NAME = "com.example.bishop.helloworld.StudentsProvider";
    static final String URL = "content://" + PROVIDER_NAME + "/students";
    static final Uri CONTENT_URI = Uri.parse(URL);

    static final String _ID = "_id";
    static final String NAME = "name";
    static final String GRADE = "grade";

    private static HashMap<String, String> STUDENTS_PROJECTION_MAP;

    static final int STUDENTS = 1;
    static final int STUDENT_ID = 2;

    static final UriMatcher uriMatcher;
    static {...}

    /**...*/

    private SQLiteDatabase db;
    static final String DATABASE_NAME = "College";
    static final String STUDENTS_TABLE_NAME = "students";
    static final int DATABASE_VERSION = 1;
    static final String CREATE_DB_TABLE =
        "CREATE TABLE " + STUDENTS_TABLE_NAME +
        " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        " name TEXT NOT NULL, " +
        " grade TEXT NOT NULL);";

    /**...*/

    private static class DatabaseHelper extends SQLiteOpenHelper {...}
}
```

```
@Override
public boolean onCreate() {...}

@Override
public Uri insert(Uri uri, ContentValues values) {...}

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

@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {...}

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

@Override
public String getType(Uri uri) {...}
}
```

# CONTENT PROVIDER

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.bishop.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <provider
            android:name=".StudentsProvider"
            android:authorities="com.example.bishop.helloworld.StudentsProvider"
            android:enabled="true"
            android:exported="true"></provider>
    </application>
</manifest>
```

# CONTENT PROVIDER

- 授权 (android:authorities): 用于在系统内标识整个提供程序的符号名称。
- 提供程序类名 ( android:name ): 实现 ContentProvider 的类。
- 启动和控制属性: 这些属性决定 Android 系统如何以及何时启动提供程序、提供程序的进程特性以及其他运行时设置:
- android:enabled: 允许系统启动提供程序的标志。
- android:exported: 允许其他应用使用此提供程序的标志。
- android:initOrder: 此提供程序相对于同一进程中其他提供程序的启动顺序。
- android:multiProcess: 允许系统在与调用客户端相同的进程中启动提供程序的标志。
- android:process: 应在其中运行提供程序的进程的名称。
- android:syncable: 指示提供程序的数据将与服务器上的数据同步的标志。

# CONTENT PROVIDER

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.bishop.helloworld.MainActivity">

    <TextView...>

    <TextView...>

    <ImageButton...>

    <Button...>

    <EditText...>

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText2"
        android:layout_alignTop="@+id/editText"
        android:layout_alignLeft="@+id/textView1"
        android:layout_alignStart="@+id/textView1"
        android:layout_alignRight="@+id/textView1"
        android:layout_alignEnd="@+id/textView1"
        android:hint="Name"
        android:textColorHint="@android:color/holo_blue_light" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText3"
        android:layout_below="@+id/editText"
        android:layout_alignLeft="@+id/editText2"
        android:layout_alignStart="@+id/editText2"
        android:layout_alignRight="@+id/editText2"
        android:layout_alignEnd="@+id/editText2"
        android:hint="Grade"
        android:textColorHint="@android:color/holo_blue_bright" />

    <Button...>
</RelativeLayout>
```

8:00

HelloWorld

Content provider

Tutorials point

Name

Grade

ADD NAME

RETRIVE STUDENT

Content provider

Tutorials point

ADD NAME

RETRIVE STUDENT

# CONTENT PROVIDER

```
package com.example.bishop.helloworld;

import ...

public class MainActivity extends Activity {

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

    public void onClickAddName(View view) {
        // Add a new student record
        ContentValues values = new ContentValues();
        values.put(StudentsProvider.NAME,
            (EditText)findViewById(R.id.editText2)).getText().toString());

        values.put(StudentsProvider.GRADE,
            (EditText)findViewById(R.id.editText3)).getText().toString());

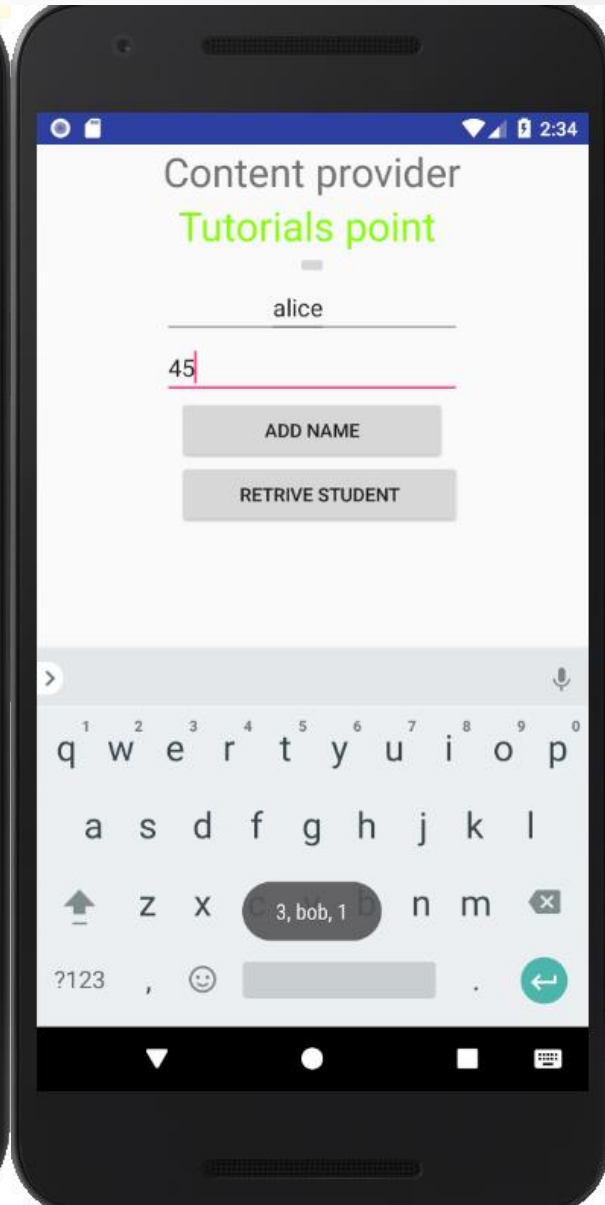
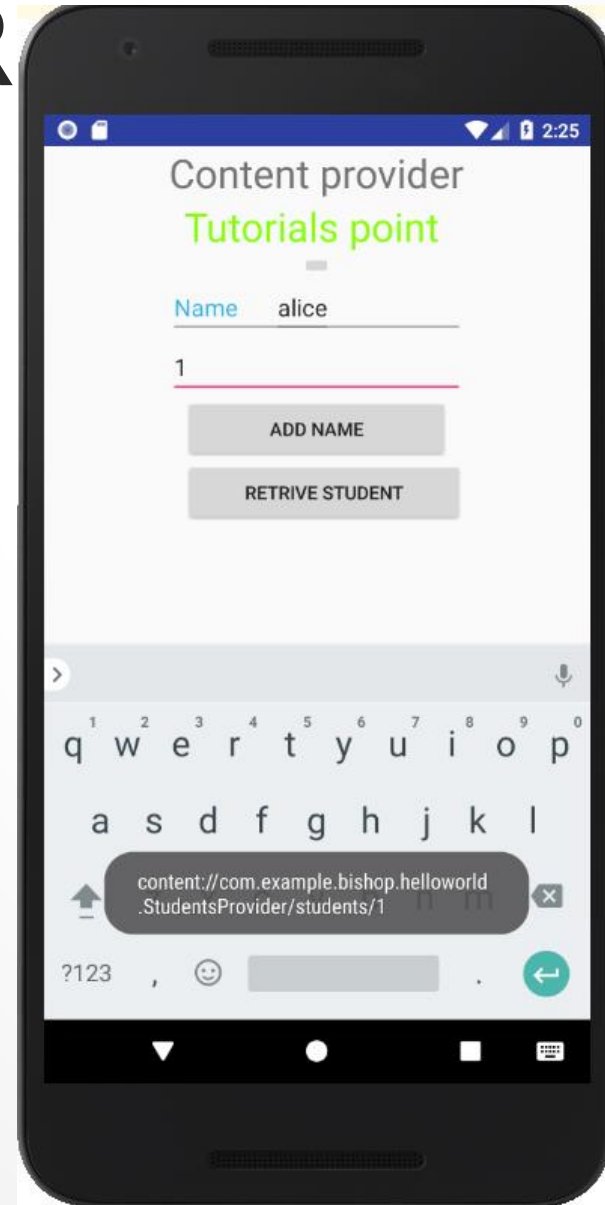
        Uri uri = getContentResolver().insert(
            StudentsProvider.CONTENT_URI, values);

        Toast.makeText(getBaseContext(),
            uri.toString(), Toast.LENGTH_LONG).show();
    }

    public void onClickRetrieveStudents(View view) {
        // Retrieve student records
        String URL = "content://com.example.bishop.helloworld.StudentsProvider";

        Uri students = Uri.parse(URL);
        Cursor c = managedQuery(students, projection: null, selection: null, selectionArgs: null, sortOrder: "name");

        if (c.moveToFirst()) {
            do {
                Toast.makeText(context: this,
                    text: c.getString(c.getColumnIndex(StudentsProvider._ID)) +
                        ", " + c.getString(c.getColumnIndex(StudentsProvider.NAME)) +
                        ", " + c.getString(c.getColumnIndex(StudentsProvider.GRADE)),
                    Toast.LENGTH_SHORT).show();
            } while (c.moveToNext());
        }
    }
}
```



# CONTENT PROVIDER

- 使用ContentResolver访问ContentProvider:
- 在AndroidManifest.xml中静态申请访问权限:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.bishop.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

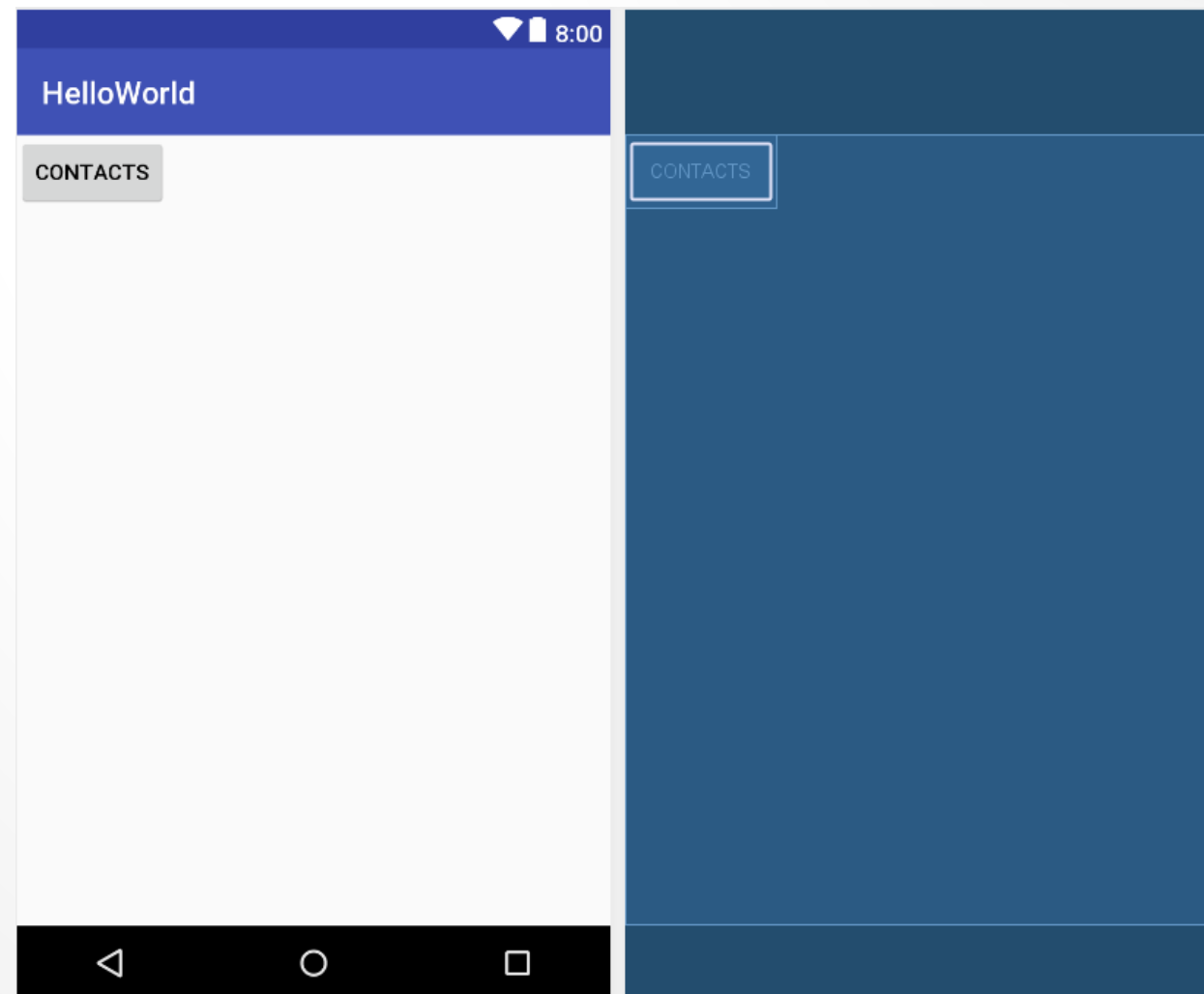
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.READ_CONTACTS" />
</manifest>
```

# CONTENT PROVIDER

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.bishop.helloworld.MainActivity">

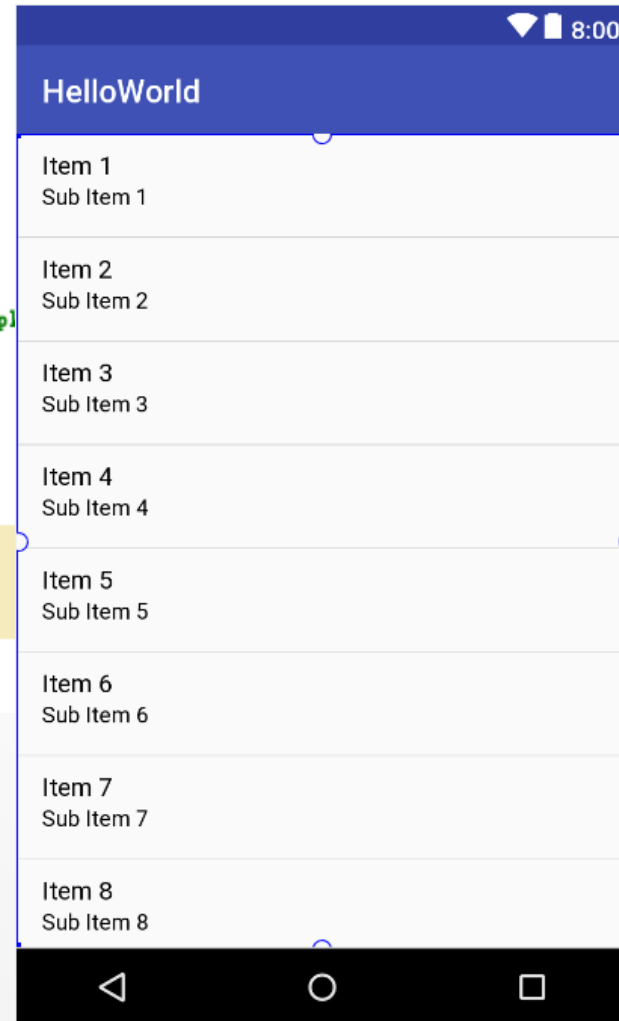
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Contacts"
        android:id="@+id/btn_contacts" />

</RelativeLayout>
```



# CONTENT PROVIDER

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.bishop.helloworld.Main2Activity">
    <ListView
        android:id="@+id/contacts_views"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"></ListView>
</android.support.constraint.ConstraintLayout>
```





# CONTENT PROVIDER

- 在Main2Activity.java中动态申请访问权限:

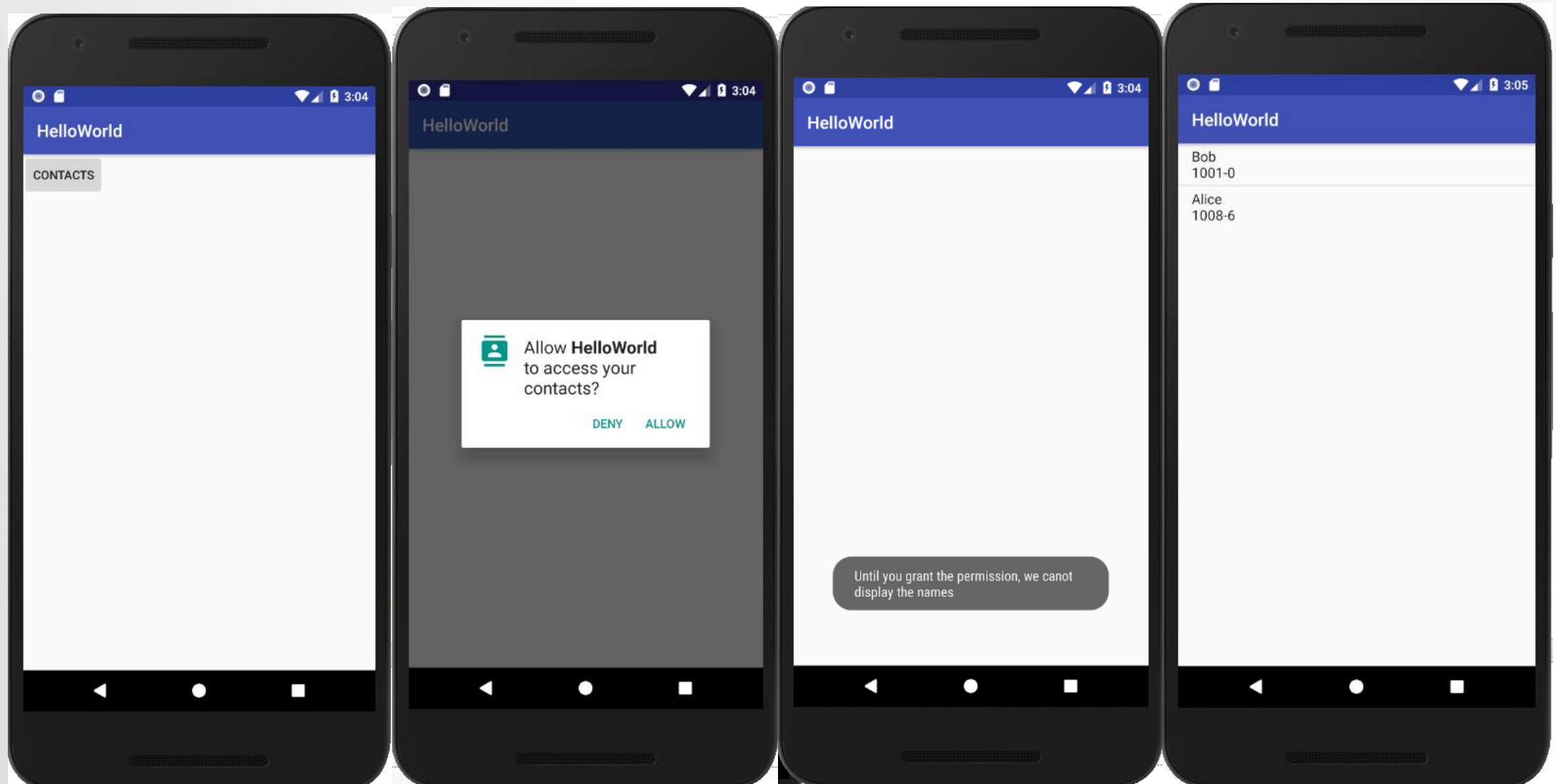
```
private void showContacts() {  
    // Check the SDK version and whether the permission is already granted or not.  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M && checkSelfPermission(Manifest.permission.READ_CONTACTS) != PackageManager.PERMISSION_GRANTED) {  
        requestPermissions(new String[]{Manifest.permission.READ_CONTACTS}, PERMISSIONS_REQUEST_READ_CONTACTS);  
        //After this point you wait for callback in onRequestPermissionsResult(int, String[], int[]) overridden method  
    } else {  
        readContacts();  
    }  
}  
  
@Override  
public void onRequestPermissionsResult(int requestCode, String[] permissions,  
                                       int[] grantResults) {  
    if (requestCode == PERMISSIONS_REQUEST_READ_CONTACTS) {  
        if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {  
            // Permission is granted  
  
            showContacts();  
        } else {  
            Toast.makeText(context, this, text: "Until you grant the permission, we cannot display the names", Toast.LENGTH_SHORT).show();  
        }  
    }  
}
```

# CONTENT PROVIDER

- 使用getContentResolver()方法读取联系人信息：

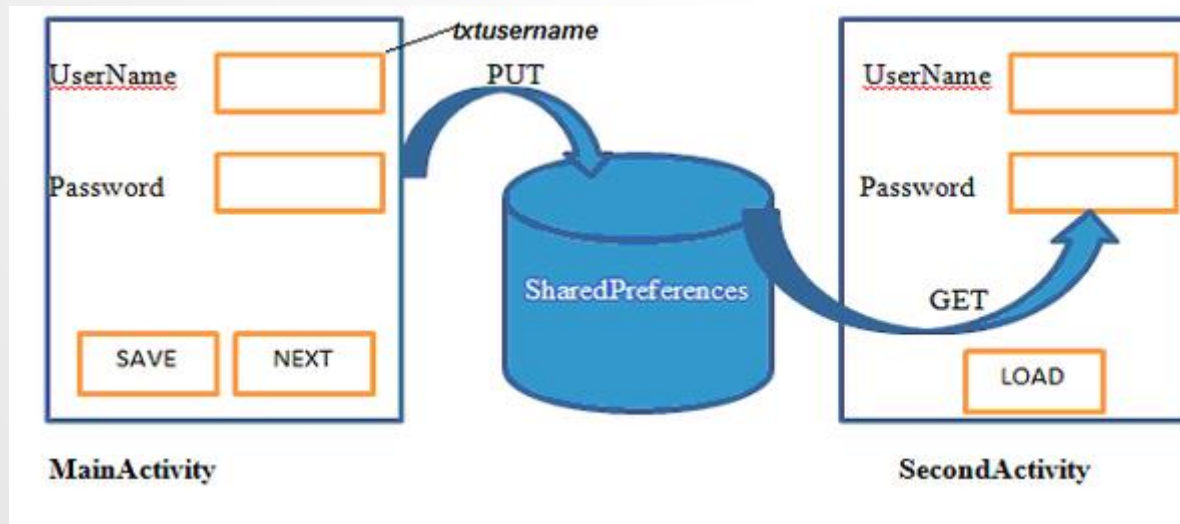
```
private void readContacts(){
    Cursor cursor=null;
    try{
        cursor=getContentResolver().query(
            ContactsContract.CommonDataKinds.Phone.CONTENT_URI, projection: null, selection: null, selectionArgs: null, sortOrder: null
        );
        while(cursor.moveToNext()){
            String displayName =cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME));
            String number=cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER));
            contactsList.add(displayName+"\n"+number);
        }
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        if(cursor!=null){
            cursor.close();
        }
    }
}
```

# CONTENT PROVIDER



# SHARED PREFERENCES

- SharedPreferences 对象指向包含键值对的文件并提供读写这些文件的简单方法。每个 SharedPreferences 文件由框架进行管理并且可以专用或共享。
- SharedPreferences是一个轻量级的存储类，特别适合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。



# SHARED PREFERENCES

- 使用 SharedPreferences 保存key-value对的步骤一般是这样：
- 使用Activity类的getSharedPreferences方法获取到SharedPreferences 对象，指定文件名和访问权限。
- 获得SharedPreferences.Editor对象，并使用该对象的 putXXX方法保存key-value对。
- 通过SharedPreferences.Editor的commit方法保存（提交） key-value对。

# SHARED PREFERENCES

- 使用getSharedPreferences(String name, int mode)方法来获取SharedPreferences实例，该方法第一个参数为实例名称（不带后缀），第二个参数为操作模式，共四种：
- Context.MODE\_PRIVATE：为默认操作模式，代表该文件是私有数据，只能被应用本身访问，在该模式下，写入的内容会覆盖原文件的内容，如果想把新写入的内容追加到原文件中。可以使用Context.MODE\_APPEND
- Context.MODE\_APPEND：模式会检查文件是否存在，存在就往文件追加内容，否则就创建新文件。
- MODE\_WORLD\_READABLE：表示当前文件可以被其他应用读取；
- MODE\_WORLD\_WRITEABLE：表示当前文件可以被其他应用写入。

# SHARED PREFERENCES

- 获取SharedPreferences实例之后，需要创建SharedPreferences.Editor以实现修改功能：
- `SharedPreferences pref = getSharedPreferences("myPref", MODE_PRIVATE);`
- `SharedPreferences.Editor editor = pref.edit();`

# SHARED PREFERENCES

- 在SharedPreferences.Editor中使用putXXX方法存储键值对，并使用commit方法提交数据：
- `editor.putString( "name" , "刘全有");`
- `editor.putInt("age", 25);`
- `editor.putLong("time", System.currentTimeMillis());`
- `editor.putBoolean("default", true);`
- ...
- `editor.commit();`



# SHARED PREFERENCES

- 使用getXXX方法访问键值对，读取SharedPreferences实例中存储的数据，该方法第二个参数为默认值，当键值对中不存在该Key时返回：
- `SharedPreferences pref = getSharedPreferences("myPref", MODE_PRIVATE);`
- `String name = pref.getString("name", "defaultName");`
- `Int age = pref.getInt("age", 0);`
- `Long time = pref.getLong("time", 0);`
- `Boolean default = pref.getBoolean("default", true);`

# SHARED PREFERENCES

- 使用remove方法删除SharedPreferences实例中某个键值对:
- `SharedPreferences pref = getSharedPreferences("myPref", MODE_PRIVATE);`
- `SharedPreferences.Editor editor = pref.edit();`
- `Editor.remove( "name" );`
- `editor.commit();`

# SHARED PREFERENCES

- 使用clear方法删除SharedPreferences实例中全部键值对:
- `SharedPreferences pref = getSharedPreferences("myPref", MODE_PRIVATE);`
- `SharedPreferences.Editor editor = pref.edit();`
- `Editor.clear();`
- `editor.commit();`

# SHARED PREFERENCES

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.bishop.helloworld.MainActivity">

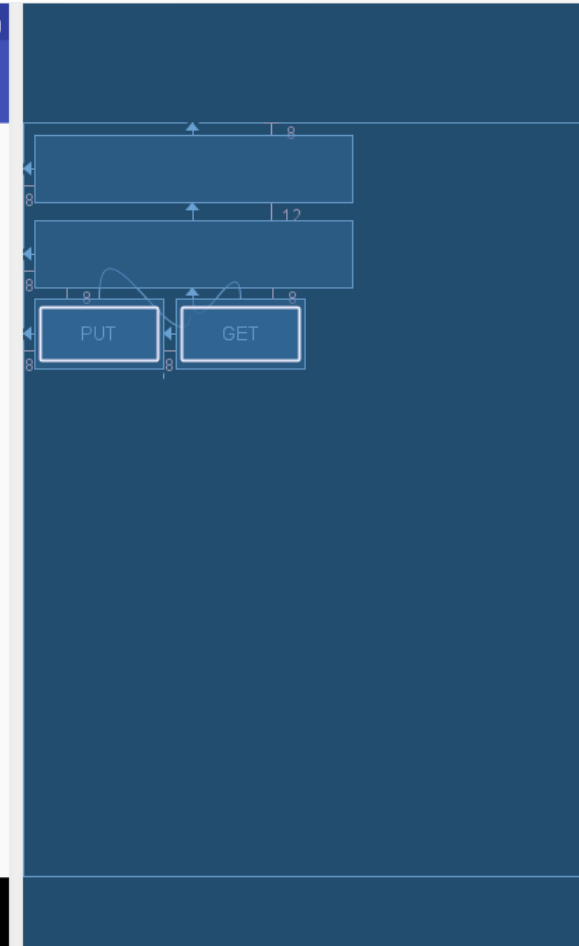
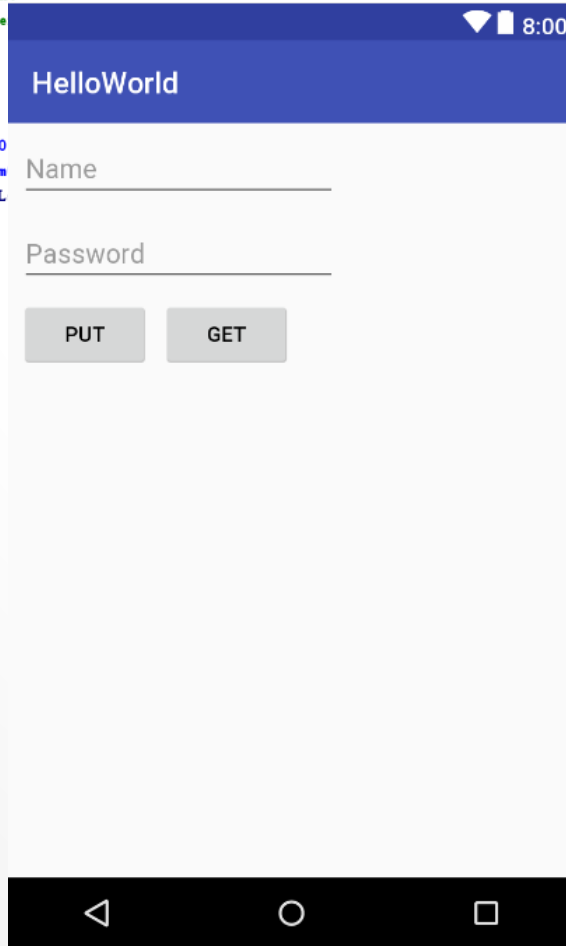
    <EditText
        android:id="@+id/editText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:inputType="textPersonName"
        android:hint="Name"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/editText2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignStart="@+id/editText"
        android:layout_below="@+id/editText"
        android:layout_marginStart="8dp"
        android:layout_marginTop="12dp"
        android:ems="10"
        android:inputType="textPersonName"
        android:hint="Password"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="put"
        android:onClick="put"
```

```
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText2" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="get"
        android:onClick="get"
        app:layout_constraintStart_toEndOf="@+id/editText2"
        app:layout_constraintTop_toBottomOf="@+id/editText2" />
</android.support.constraint.ConstraintLayout>
```

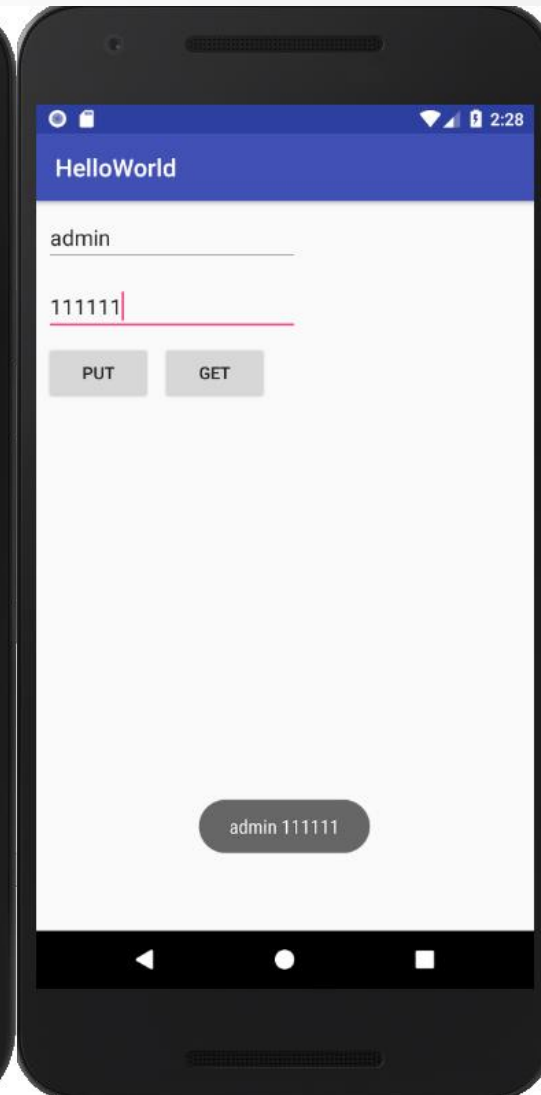
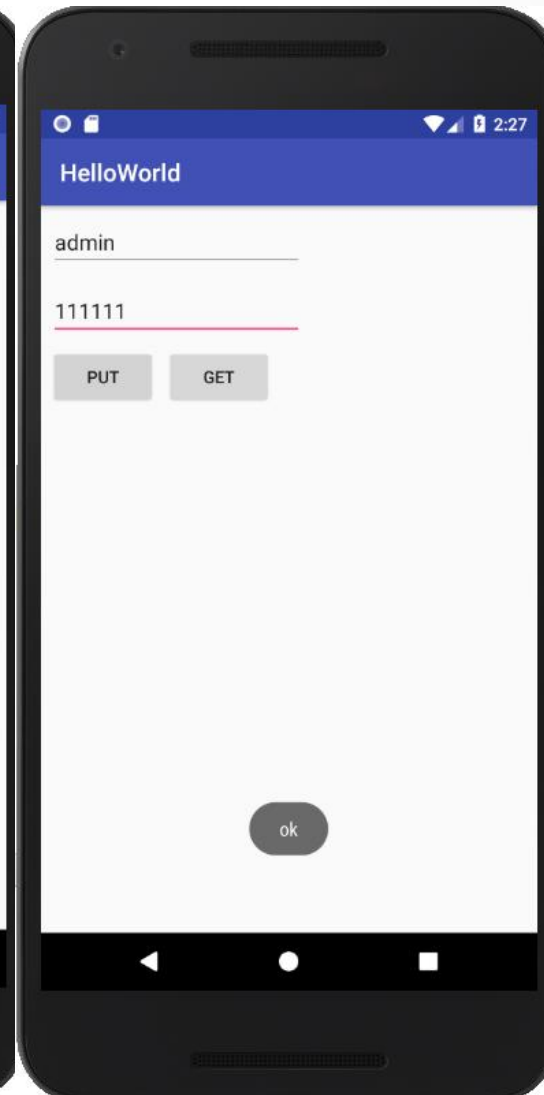
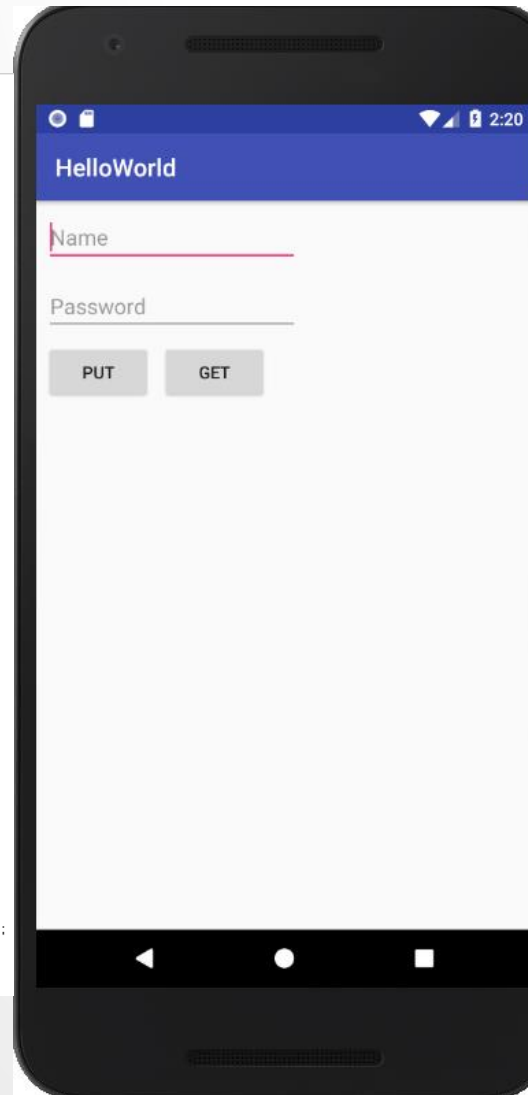


# SHARED PREFERENCES

```
package com.example.bishop.helloworld;

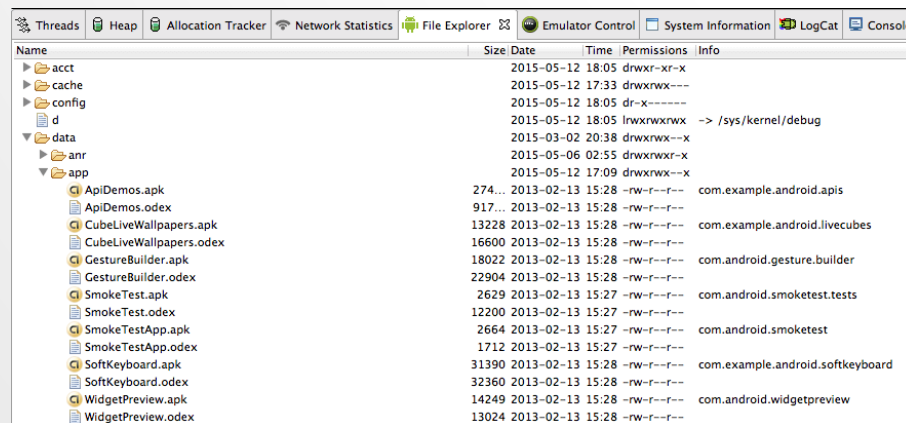
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    EditText name;
    EditText password;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        name=(EditText)findViewById(R.id.editText);
        password=(EditText)findViewById(R.id.editText2);
    }
    public void put(View view) {
        SharedPreferences pref = super.getSharedPreferences( name: "myPref", MODE_PRIVATE);
        SharedPreferences.Editor editor = pref.edit();
        editor.putString("name", name.getText().toString());
        editor.putString("password", password.getText().toString());
        editor.commit();
        Toast.makeText( context: MainActivity.this, text: "ok",Toast.LENGTH_LONG).show();
    }
    public void get(View view) {
        SharedPreferences pref = super.getSharedPreferences( name: "myPref", MODE_PRIVATE);
        String name=pref.getString( key: "name", defValue: "chosen one");
        String password=pref.getString( key: "password", defValue: "123456");
        Toast.makeText( context: MainActivity.this, text: name+" "+password,Toast.LENGTH_LONG).show();
    }
}
```



# FILE

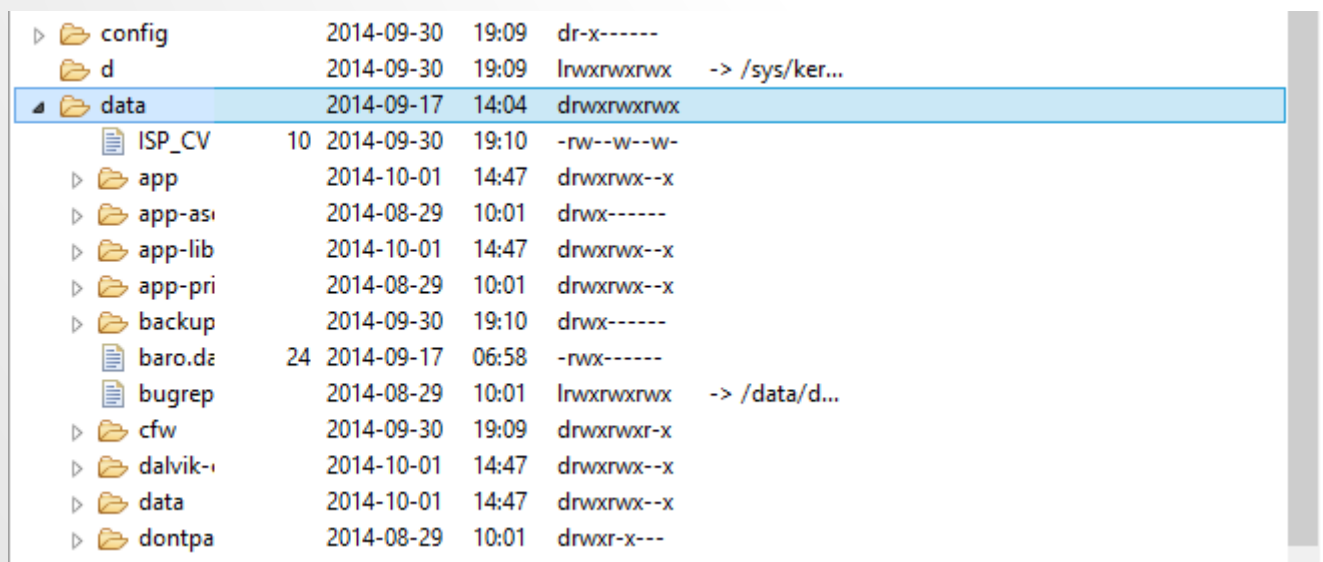
- Android应用可能访问以下文件系统路径：
- /data/app：该文件夹存放着系统中安装的第三方应用的 apk 文件。Android 中应用的安装就是将应用的安装包原封不动地拷贝到 /data/app 目录下，
- 每个应用安装包本质上就是一个 zip 格式的压缩文件。为了提升应用的启动效率，Android 会将解压出来的 dex 格式的应用代码文件解析提取后，缓存在 /data/dalvik-cache 目录下。



Name	Size	Date	Time	Permissions	Info
▶ acct		2015-05-12	18:05	drwxr-xr-x	
▶ cache		2015-05-12	17:33	drwxrwx---	
▶ config		2015-05-12	18:05	dr-x-----	
▶ d		2015-05-12	18:05	lrwxrwxrwx	-> /sys/kernel/debug
▼ data		2015-03-02	20:38	drwxrwx--x	
▶ anr		2015-05-06	02:55	drwxrwxr-x	
▼ app		2015-05-12	17:09	drwxrwx--x	
ApiDemos.apk	274...	2013-02-13	15:28	-rw-r--r--	com.example.android.apis
ApiDemos.odex	917...	2013-02-13	15:28	-rw-r--r--	
CubeLiveWallpapers.apk	13228	2013-02-13	15:28	-rw-r--r--	com.example.android.livecubes
CubeLiveWallpapers.odex	16600	2013-02-13	15:28	-rw-r--r--	
GestureBuilder.apk	18022	2013-02-13	15:28	-rw-r--r--	com.android.gesture.builder
GestureBuilder.odex	22904	2013-02-13	15:28	-rw-r--r--	
SmokeTest.apk	2629	2013-02-13	15:27	-rw-r--r--	com.android.smoketest.tests
SmokeTest.odex	12200	2013-02-13	15:27	-rw-r--r--	
SmokeTestApp.apk	2664	2013-02-13	15:27	-rw-r--r--	com.android.smoketest
SmokeTestApp.odex	1712	2013-02-13	15:27	-rw-r--r--	
SoftKeyboard.apk	31390	2013-02-13	15:28	-rw-r--r--	com.example.android.softkeyboard
SoftKeyboard.odex	32360	2013-02-13	15:28	-rw-r--r--	
WidgetPreview.apk	14249	2013-02-13	15:28	-rw-r--r--	com.android.widgetpreview
WidgetPreview.odex	13024	2013-02-13	15:28	-rw-r--r--	

# FILE

- /data/data: 该文件夹存放存储包私有数据，对于设备中每一个安装的 App，系统都会在内部存储空间的 data/data 目录下以应用包名为名字自动创建与之对应的文件夹。
- 用户卸载 App 时，系统自动删除 data/data 目录下对应包名的文件夹及其内容。



The screenshot shows a file manager interface with a list of files and folders. The 'data' folder is selected and highlighted in blue. The list includes folders like 'config', 'd', 'data', 'app', 'app-as', 'app-lib', 'app-pri', 'backup', 'baro.da', 'bugrep', 'cfw', 'dalvik-', 'data', and 'dontpa'. Each entry shows its name, size, date, time, and permissions. Some entries also show symbolic links like '-> /sys/ker...' and '-> /data/d...'.

Name	Size	Date	Time	Permissions	Links
config		2014-09-30	19:09	dr-x-----	
d		2014-09-30	19:09	lrwxrwxrwx	-> /sys/ker...
data		2014-09-17	14:04	drwxrwxrwx	
ISP_CV	10	2014-09-30	19:10	-rw--w--w-	
app		2014-10-01	14:47	drwxrwx--x	
app-as		2014-08-29	10:01	drwx-----	
app-lib		2014-10-01	14:47	drwxrwx--x	
app-pri		2014-08-29	10:01	drwxrwx--x	
backup		2014-09-30	19:10	drwx-----	
baro.da	24	2014-09-17	06:58	-rwx-----	
bugrep		2014-08-29	10:01	lrwxrwxrwx	-> /data/d...
cfw		2014-09-30	19:09	drwxrwxr-x	
dalvik-		2014-10-01	14:47	drwxrwx--x	
data		2014-10-01	14:47	drwxrwx--x	
dontpa		2014-08-29	10:01	drwxr-x---	

# FILE

- /system: 通过Environment.getRootDirectory() 访问, 该目录下也有一个 app 目录, 存放的是系统应用的 apk 文件。
- /system/app 和 /data/app 的区别为:
- /system/app 里的软件获取了所有权限。
- /system/app 只能 root 后删除。
- /system/app 文件夹有大小限制, 卸载/system/app 目录下的文件并不会增加系统空间, 即可用 ROM 空间。



# FILE

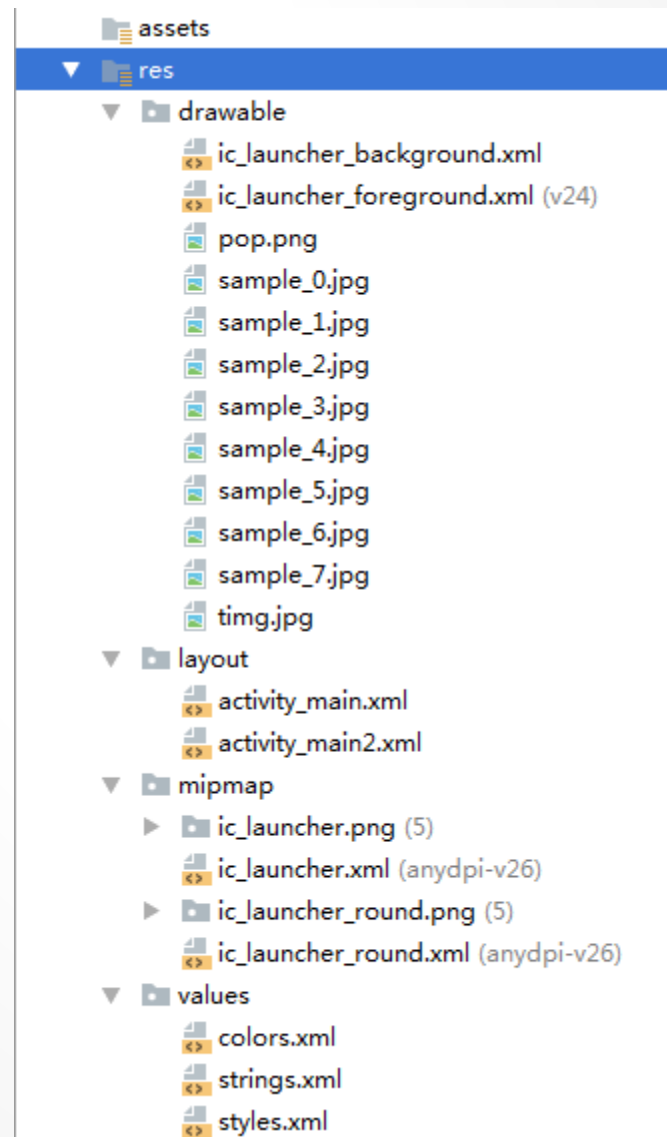
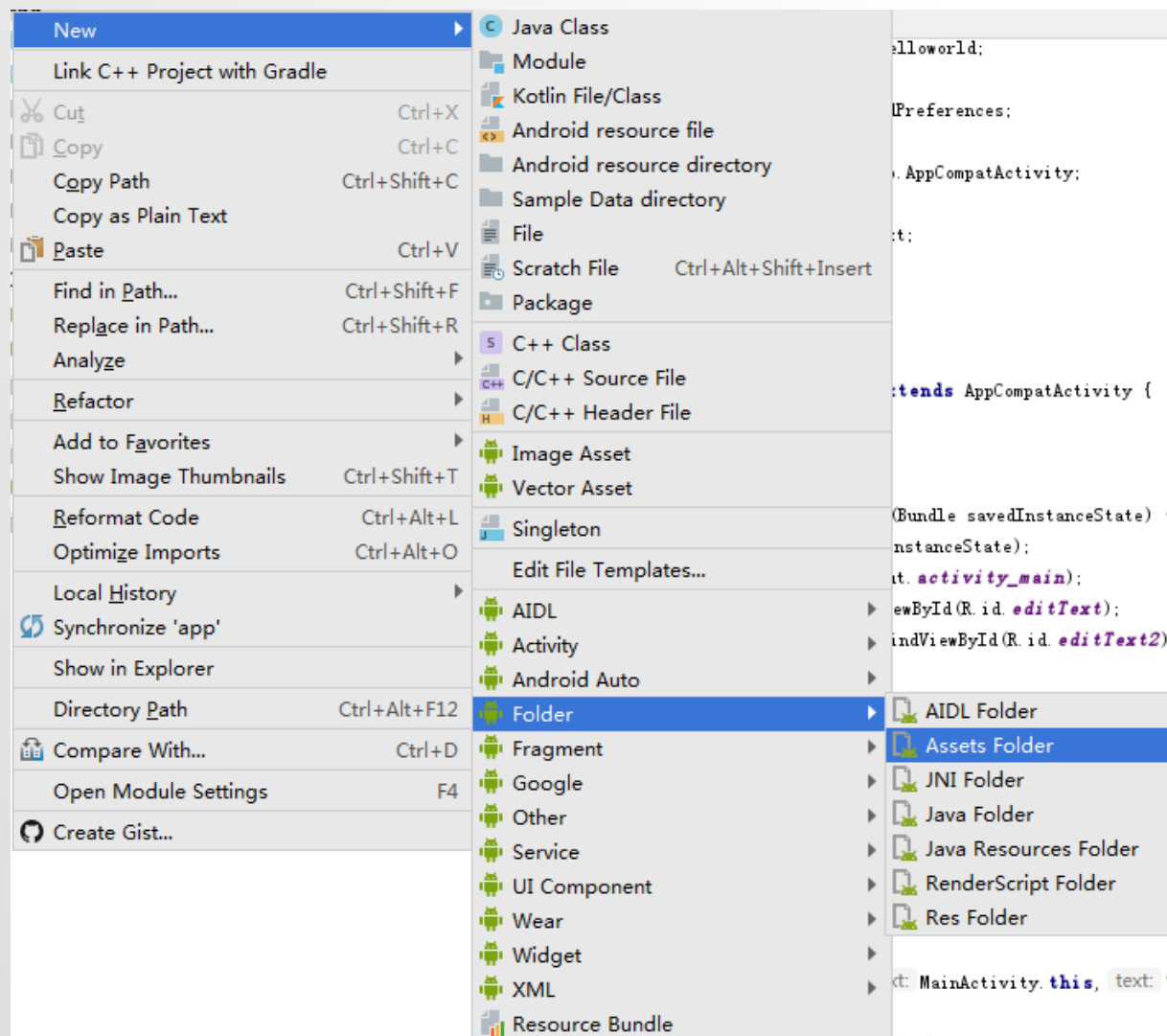
- /mnt : 这个目录专门用来当作挂载点挂在外部设备, 如 SD 卡, 将会被系统视作一个文件夹, 这个文件夹将会被系统嵌入到文件系统的 mnt 目录中, 所以在 /mnt 目录下也会看到一个 sdcard 的快捷方式:

init.trace.rc	1795	1970-01-01	08:00	-rwxr-x---	
init.usb.rc	3915	1970-01-01	08:00	-rwxr-x---	
init.wlan.rc	21198	1970-01-01	08:00	-rwxr-x---	
mem_proc_slab.sh	1733	1970-01-01	08:00	-rw-r--r--	
mnt					
UDiskA		1970-03-20	16:53	drwxrwxrwx	
UDiskB		1970-03-20	16:53	lrwxrwxrwx	-> /storage/UDiskA
UDiskC		1970-03-20	16:53	lrwxrwxrwx	-> /storage/UDiskB
UDiskD		1970-03-20	16:53	lrwxrwxrwx	-> /storage/UDiskC
UDiskE		1970-03-20	16:53	lrwxrwxrwx	-> /storage/UDiskD
UDiskF		1970-03-20	16:53	lrwxrwxrwx	-> /storage/UDiskE
asec		1970-03-20	16:53	drwxrwxrwx	-> /storage/UDiskF
obb		1970-03-20	16:53	drwxrwxrwx	
sdcard		1970-03-20	16:53	lrwxrwxrwx	-> /storage/emulated/legacy
secure		1970-03-20	16:53	drwxrwxrwx	
shell		1970-03-20	16:53	drwxrwxrwx	
persist		1970-01-01	08:00	drwxrwx--x	
proc		1970-01-01	08:00	dr-xr-xr-x	
property_contexts					
res	2109	1970-01-01	08:00	-rw-r--r--	
root		1970-01-01	08:00	drwxr-xr-x	
sbin		2015-06-03	16:39	drwx-----	
sdcard		1970-01-01	08:00	drwxr-x---	
sdcard2		1970-03-20	16:53	lrwxrwxrwx	-> /storage/emulated/legacy
seapp_contexts		1970-03-20	16:53	lrwxrwxrwx	-> /storage/sdcard1
sepolicy	611	1970-01-01	08:00	-rw-r--r--	
...	63573	1970-01-01	08:00	...	

# FILE

- 在Android APK中，除了被编译的代码以外，还可以允许存储两种资源文件：
- res：文件会被映射到R.java文件中，访问的时候直接通过资源ID即可访问，而且不能有目录结构，即不能再创建文件夹。
- assets：不会映射到R.java文件中，通过AssetManager来访问，能有目录结构，即可以自行创建文件夹。
- 注意：res和assets只能读取，不能修改，每个文件大小不能超过1MB。

# FILE



# FILE

目录	资源类型
<code>animator/</code>	用于定义 <a href="#">属性动画</a> 的 XML 文件。
<code>anim/</code>	定义 <a href="#">渐变动画</a> 的 XML 文件。（属性动画也可以保存在此目录中，但是为了区分这两种类型，属性动画首选 <code>animator/</code> 目录。）
<code>color/</code>	用于定义颜色状态列表的 XML 文件。请参阅 <a href="#">颜色状态列表资源</a>
<code>drawable/</code>	<p>位图文件（<code>.png</code>、<code>.9.png</code>、<code>.jpg</code>、<code>.gif</code>）或编译为以下可绘制对象资源子类型的 XML 文件：</p> <ul style="list-style-type: none"><li>• 位图文件</li><li>• 九宫格（可调整大小的位图）</li><li>• 状态列表</li><li>• 形状</li><li>• 动画可绘制对象</li><li>• 其他可绘制对象</li></ul> <p>请参阅 <a href="#">可绘制对象资源</a>。</p>
<code>mipmap/</code>	适用于不同启动器图标密度的可绘制对象文件。如需了解有关使用 <code>mipmap/</code> 文件夹管理启动器图标的详细信息，请参阅 <a href="#">管理项目概览</a> 。
<code>layout/</code>	用于定义用户界面布局的 XML 文件。 请参阅 <a href="#">布局资源</a> 。
<code>menu/</code>	用于定义应用菜单（如选项菜单、上下文菜单或子菜单）的 XML 文件。请参阅 <a href="#">菜单资源</a> 。

# FILE

raw/	<p>要以原始形式保存的任意文件。要使用原始 <code>InputStream</code> 打开这些资源，请使用资源 ID（即 <code>R.raw.filename</code>）调用 <code>Resources.openRawResource()</code>。</p> <p>但是，如需访问原始文件名和文件层次结构，则可以考虑将某些资源保存在 <code>assets/</code> 目录下（而不是 <code>res/raw/</code>）。<code>assets/</code> 中的文件没有资源 ID，因此您只能使用 <code>AssetManager</code> 读取这些文件。</p>
values/	<p>包含字符串、整型数和颜色等简单值的 XML 文件。</p> <p>其他 <code>res/</code> 子目录中的 XML 资源文件是根据 XML 文件名定义单个资源，而 <code>values/</code> 目录中的文件可描述多个资源。对于此目录中的文件，<code>&lt;resources&gt;</code> 元素的每个子元素均定义一个资源。例如，<code>&lt;string&gt;</code> 元素创建 <code>R.string</code> 资源，<code>&lt;color&gt;</code> 元素创建 <code>R.color</code> 资源。</p> <p>由于每个资源均用其自己的 XML 元素定义，因此您可以根据自己的需要命名文件，并将不同的资源类型放在一个文件中。但是，为了清晰起见，您可能需要将独特的资源类型放在不同的文件中。例如，对于可在此目录中创建的资源，下面给出了相应的文件名约定：</p> <ul style="list-style-type: none"><li>arrays.xml，用于资源数组（<a href="#">类型化数组</a>）。</li><li>colors.xml：颜色值。</li><li>dimens.xml：尺寸值。</li><li>strings.xml：字符串值。</li><li>styles.xml：样式。</li></ul> <p>请参阅<a href="#">字符串资源</a>、<a href="#">样式资源</a>和<a href="#">更多资源类型</a>。</p>
xml/	<p>可以在运行时通过调用 <code>Resources.getXML()</code> 读取的任意 XML 文件。各种 XML 配置文件（如<a href="#">可搜索配置</a>）都必须保存在此处。</p>

# FILE

- 读res资源文件:
- `InputStream is = getResources().openRawResource(R.drawable.pop);`
- 读assets资源文件:
- `AssetManager am = getAssets();`
- `InputStream is = am.open("filename");`

# FILE

- 对于设备中每一个安装的 App，系统都会在内部存储空间的 data/data 目录下以应用包名为名字自动创建与之对应的文件夹。
- 用户卸载 App 时，系统自动删除 data/data 目录下对应包名的文件夹及其内容。该目录下又把存储内容进行了分类：
- data/data/包名/cache： 存放的 APP 的缓存信息。
- data/data/包名/databases： 存放 APP 的数据库信息。
- data/data/包名/files： 存放 APP 的文件信息。
- data/data/包名/shared\_prefs： 存放 APP 内的 SharedPreferences。


# FILE

- 若需要访问内部存储器data/data/包名/路径上的资源文件，可使用FileOutputStream和FileInputStream。
- 若需要访问SD卡存储空间，需要先获取READ\_EXTERNAL\_STORAGE或WRITE\_EXTERNAL\_STORAGE系统权限。
- 外部存储在 Android 文件系统中是 sdcard 目录，这里只是一个快捷方式，真正的目录是 /storage/emulated/legacy 文件夹。
- 仅能对data/data/包名/路径上的资源文件使用openFileOutput和openFileInput两个方法，不可进行其他操作。



# FILE

- 在AndroidManifest.xml中获取SD卡文件写入、创建与删除权限：



The screenshot displays the Android Studio environment. On the left, the 'Project' tool window shows a file tree with the 'sdcard' directory highlighted under the 'storage' folder. On the right, the 'Manifest' tool window shows the content of the `AndroidManifest.xml` file. The file starts with the XML declaration and the `manifest` tag, followed by the `package` attribute set to `com.example.bishop.helloworld`. Two `uses-permission` tags are present, requesting `android.permission.MOUNT_FORMAT_FILESYSTEMS` and `android.permission.WRITE_EXTERNAL_STORAGE`. The `application` tag includes attributes for `allowBackup`, `icon`, `label`, `roundIcon`, `supportsRtl`, and `theme`. An `activity` tag for `.MainActivity` is also shown, with an `intent-filter` for the `android.intent.action.MAIN` action and the `android.intent.category.LAUNCHER` category.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.bishop.helloworld">
    <uses-permission android:name="android.permission.MOUNT_FORMAT_FILESYSTEMS"></uses-permission>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloWorld"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

# FILE

- Environment 常用方法：
- `getDataDirectory()`：获取 Android 数据目录。
- `getDownloadCacheDirectory()`：获取 Android 下载/缓存内容目录。
- `getExternalStorageDirectory()`：获取外部存储目录即 SDCard。
- `getExternalStorageState()`：获取外部存储设备的当前状态。
- `getRootDirectory()`：获取 Android 的根目录。

# FILE

- `getExternalStorageState()`: 获取SD卡存储状态，返回值可能为以下一种：
- `MEDIA_BAD_REMOVAL`: 在没有挂载前存储媒体已经被移除。
- `MEDIA_CHECKING`: 正在检查存储媒体。
- `MEDIA_MOUNTED`: 存储媒体已经挂载，并且挂载点可读/写。
- `MEDIA_MOUNTED_READ_ONLY`: 存储媒体已经挂载，挂载点只读。
- `MEDIA_NOFS`: 存储媒体是空白或是不支持的文件系统。
- `MEDIA_REMOVED`: 存储媒体被移除。
- `MEDIA_SHARED`: 存储媒体正在通过USB共享。
- `MEDIA_UNMOUNTABLE`: 存储媒体无法挂载。
- `MEDIA_UNMOUNTED`: 存储媒体没有挂载。

# FILE

- `getExternalStoragePublicDirectory(String type)`: 提供十个公共目录用来存储相对应的文件, 返回值如下:
- `DIRECTORY_MUSIC`: `/storage/emulated/0/Music`
- `DIRECTORY_PODCASTS`: `/storage/emulated/0/Podcasts`
- `DIRECTORY_RINGTONES`: `/storage/emulated/0/Ringtones`
- `DIRECTORY_ALARMS`: `/storage/emulated/0/Alarms`
- `DIRECTORY_NOTIFICATIONS`: `/storage/emulated/0/Notifications`
- `DIRECTORY_PICTURES`: `/storage/emulated/0/Pictures`
- `DIRECTORY_MOVIES`: `/storage/emulated/0/Movies`
- `DIRECTORY_DOWNLOADS`: `/storage/emulated/0/Downloads`
- `DIRECTORY_DCIM`: `/storage/emulated/0/Dcim`
- `DIRECTORY_DOCUMENTS`: `/storage/emulated/0/Documents`

# FILE

- Android2.2 引入了基于扩展存储器的应用缓存目录，该目录指向大容量的扩展存储器。与应用的内存私有目录一样，缓存目录会随着应用的卸载一并删除。
- 和内部存储一样，会在 SD 卡的 Android/data 目录下生成对应包名的文件夹：
- `getExternalFilesDir(type)`：返回  
`/storage/emulated/0/Android/data/包名/files`。
- `getExternalCacheDir()`：返回  
`/storage/emulated/0/Android/data/应用包名/cache`。
- `getObbDir()`：返回 `/storage/emulated/0/Android/obb/包名`。

# FILE

- 对象实例化:
- `File file = new File ("/mnt/sdcard/test.txt");`
- `File file = new File("/mnt/sdcard/temp", "test.txt");`
- 判断文件是否存在:
- `file.exists();`
- 删除文件:
- `file.delete();`

# FILE

- 创建文件夹:
- `file=new File("/mnt/sdcard/temp");`
- `file.mkdir();`
- 创建文件:
- `File file = new File ("/mnt/sdcard/temp/test.txt");`
- `file.createNewFile();`

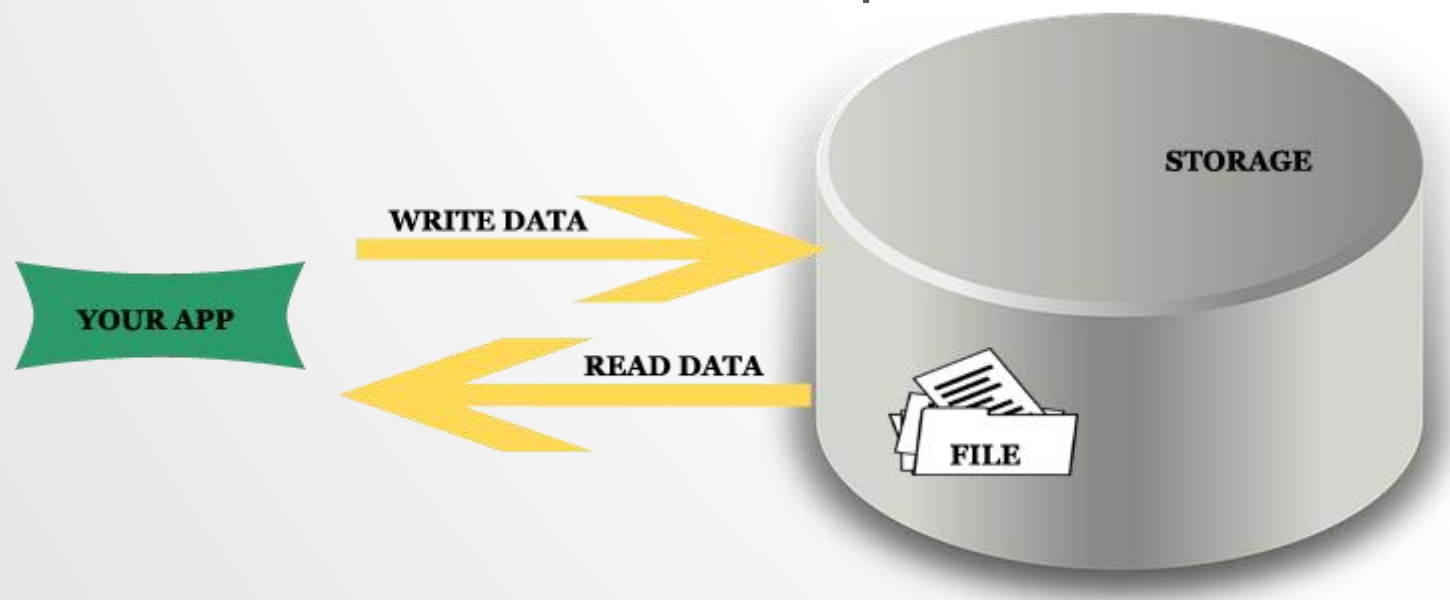
# FILE

- `boolean isDirectory()` : 测试此File对象表示的文件是否是目录。
- `boolean mkdirs()` : 创建包括父目录的目录。
- `String getAbsolutePath()`: 返回此对象表示的文件的绝对路径名。
- `String getName()` : 返回此对象表示的文件的名称。
- `String getParent()` : 返回此File对象的路径名的上一级,若路径名没有上一级, 则返回null。



# FILE

- `openFileOutput(String name,int mode);`打开应用程序私有目录下的指定私有文件以写入数据，返回一个`FileOutputStream`对象，如果文件不存在就创建这个文件。
- `openFileInput(String fileName);` 打开应用程序私有目录下的指定私有文件以读入数据，返回一个`FileInputStream`对象。



# FILE

- `openFileOutput()`方法的第一参数用于指定文件名称，不能包含路径分隔符“/”，如果文件不存在，Android 会自动创建它。
- `openFileOutput()`方法的第二参数用于指定操作模式，有四种模式：
- `Context.MODE_PRIVATE`：为默认操作模式，代表该文件是私有数据，只能被应用本身访问，在该模式下，写入的内容会覆盖原文件的内容。
- `Context.MODE_APPEND`：模式会检查文件是否存在，存在就往文件追加内容，否则就创建新文件。
- `MODE_WORLD_READABLE`：表示当前文件可以被其他应用读取。
- `MODE_WORLD_WRITEABLE`：表示当前文件可以被其他应用写入。
- `MODE_WORLD_READABLE + MODE_WORLD_WRITEABLE`：既可以读又可以写。

# FILE

- 写文件:
- `FileOutputStream fos =  
openFileOutput(fileName,MODE_PRIVATE);`
- `byte[] bytes = message.getBytes();`
- `fos.write(bytes);`
- `fos.close();`

# FILE

- 读文件:
- `FileInputStream fin = openFileInput(fileName);`
- `int length = fin.available();`
- `byte[] buffer = new byte[length];`
- `fin.read(buffer);`
- `result = EncodingUtils.getString(buffer,ENCODING);`
- `fin.close();`

THE END.