



上海建桥学院

SHANGHAI JIAN QIAO UNIVERSITY



Activity

主讲：万永权

学习目标/Target



熟悉Activity生命周期的方法，能够解释每个方法的作用

掌握Activity的创建、配置、开启和关闭的方式，能够完成创建、配置、开启和关闭Activity

熟悉Intent和IntentFilter的内容，能够归纳Intent与IntentFilter的用法

熟悉Activity的任务栈和四种启动模式的内容，能够归纳任务栈和四种启动模式的作用

掌握Activity之间的跳转方式，能够独立实现Activity之间的跳转功能

掌握Fragment的使用，能够在Activity中添加Fragment



// 主要内容

01

Activity的生命周期

02

Activity的创建、配置、启动和关闭

03

Intent与IntentFilter

04

Activity之间的跳转

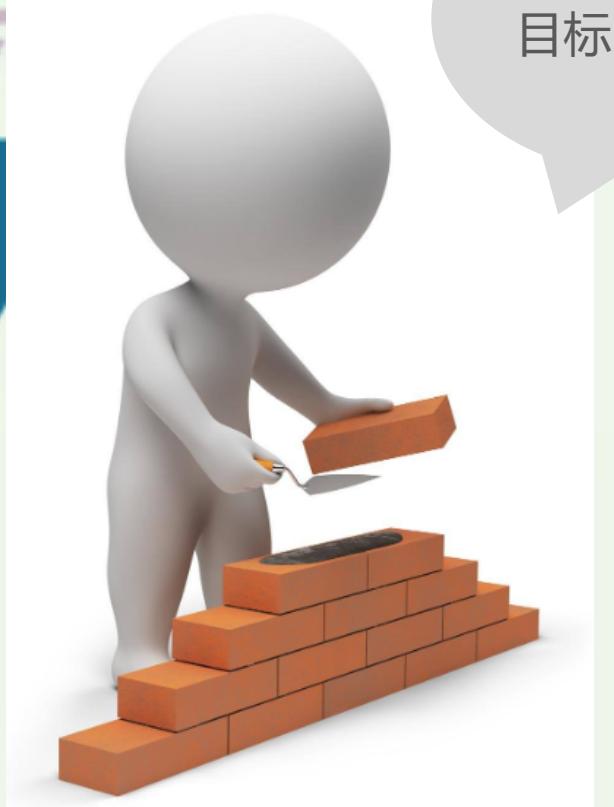
05

任务栈和启动模式



Activity的生命周期

// Activity的生命周期



先定一个小
目标！

- 熟悉 Activity 生命周期的状态，能够归纳 Activity 生命周期的5种状态
- 熟悉 Activity 生命周期的方法，能够归纳 Activity 生命周期的7个方法

// 生命周期状态

每个人都是有生命的，在每个人出生到老去的过程中需要经历**幼儿期、少年期、青春期、成年期和老年期**5个阶段。



// 生命周期状态

Activity与人一样也是有“生命”的，Activity从创建到销毁的整个过程就是Activity的生命周期，Activity的生命周期包含5种状态，这5种状态好比人类生命过程中经历的5个阶段。

启动状态

Activity的启动状态很短暂。当Activity启动之后便会进入运行状态。

暂停状态

Activity仍然可见，但无法获取焦点，用户对它操作没有响应。

销毁状态

Activity将被清理出内存。

运行状态

Activity处于屏幕最前端，可与用户进行交互。

停止状态

Activity完全不可见，系统内存不足时会销毁该Activity。



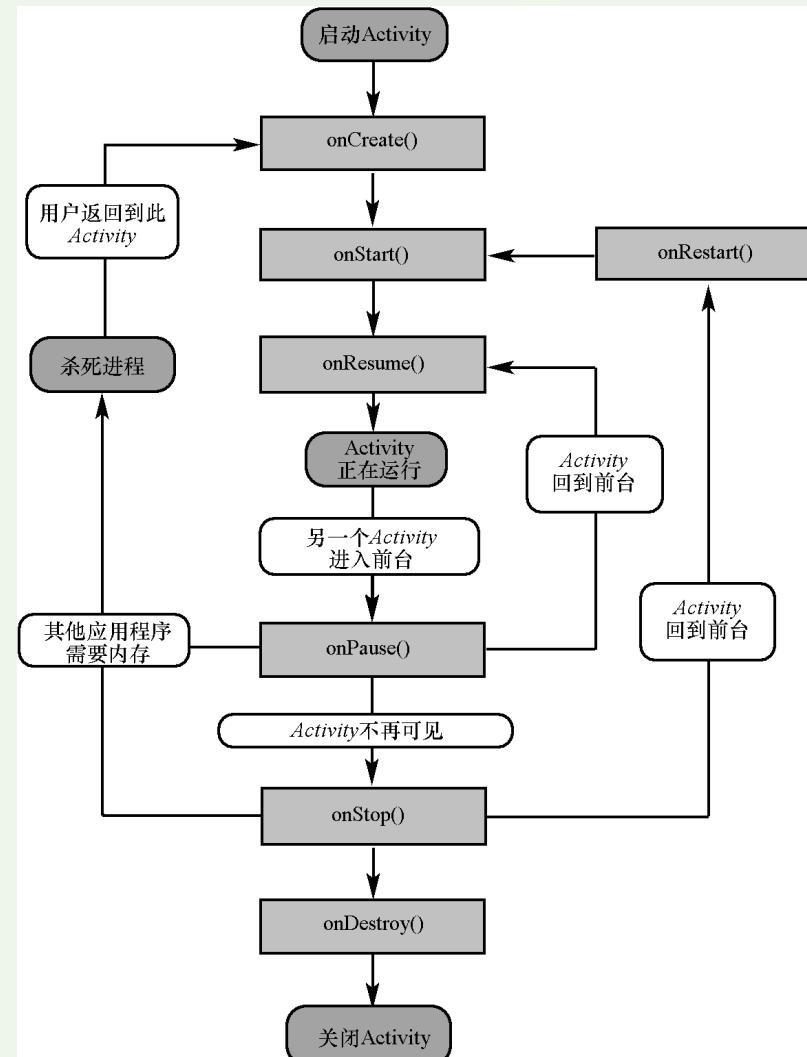
// 生命周期方法

Activity的**生命周期**包括创建、可见、获取焦点、失去焦点、不可见、重新可见、销毁等环节，针对每个环节Activity都定义了相关的回调**方法**，Activity中的回调方法具体如下。

- (1) `onCreate()` : Activity**创建时**调用，通常做一些初始化设置。
- (2) `onStart()` : Activity**即将可见时**调用。
- (3) `onResume()` : Activity**获取焦点时**调用。
- (4) `onPause()` : 当前Activity被其他Activity**覆盖或屏幕锁屏时**调用。
- (5) `onStop()` : Activity对用户**不可见时**调用。
- (6) `onRestart()` : Activity从停止状态到**再次启动时**调用。
- (7) `onDestroy()` : Activity**销毁时**调用。

// 生命周期方法

为了帮助开发者更好地理解Activity的生命周期，Google公司提供了Activity的生命周期模型，如下图所示。



// 生命周期方法

第一次运行程序时：

调用的生命周期方法为：onCreate() → onStart() → onResume()。

退出程序时：

调用的生命周期方法为：onPause() → onStop() → onDestory()。



">>>> 脚下留心



横 竖 屏 切 换 时 Activity的生命周期

当手机横竖屏切换时，程序会根据AndroidManifest.xml文件中Activity的configChanges属性值的不同而调用相应的生命周期方法。

(1) 没有设置configChanges属性的值

- 当由竖屏切换为横屏时，调用的方法依次是onPause()、onStop()、onDestory()、onCreate()、onStart()和onResume()的方法。

(2) 设置configChanges属性的值

```
<activity android:name=".MainActivity"  
         android:configChanges="orientation|keyboardHidden" />
```

打开程序时同样会依次调用onCreate()、onStart()、onResume()方法，但是当进行横竖屏切换时不会再执行其他的生命周期方法。

如果希望某一个界面一直处于竖屏或者横屏状态，并且此状态不随手机的晃动而改变，此效果可以通过在清单文件中设置Activity的screenOrientation属性来实现。

竖屏：android:screenOrientation="portrait"

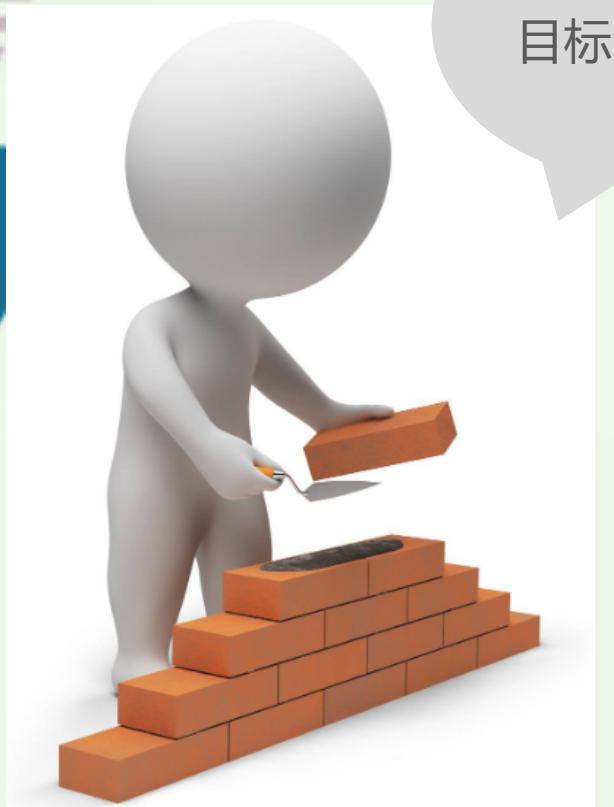
横屏：android:screenOrientation="landscape"



4.2

Activity的创建、配置、启动和关闭

4.2 Activity的创建、配置、启动和关闭

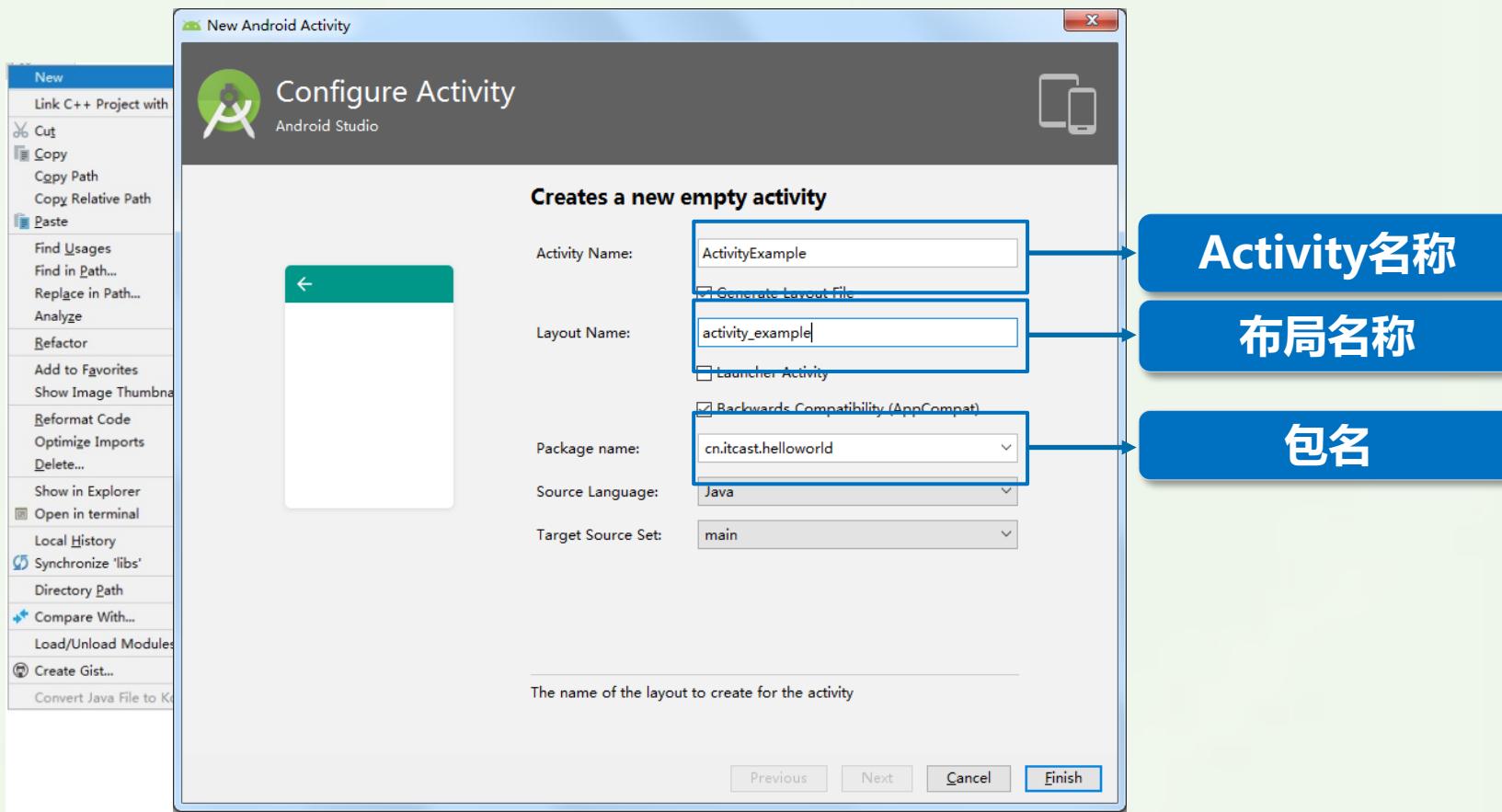


先定一个小
目标！

掌握Activity的创建、配置、启动和关闭方式，
能够完成创建、配置、开启和关闭Activity

4.2.1 创建Activity

选中程序的包名右击选择【New】→【Activity】→【Empty Activity】选项，填写Activity信息，完成创建。





4.2.2 配置Activity

在Android程序中，创建Activity可以使用Java类继承Activity的方式实现。使用此种方式创建Activity时，需要在清单文件的<application>标签中配置Activity。

```
<activity
```

```
    android:name="com.<yourname>.activitybasic.SecondActivity" />
```

如果不配置Activity，则运行程序时，程序会抛出运行时异常。

```
09-21 09:38:14.146 14893-14893/cn.itcast.activitybasic E/AndroidRuntime: java.lang.RuntimeException:  
Unable to start activity ComponentInfo{cn.itcast.activitybasic/cn.itcast.activitybasic.ActivityExample}:  
android.content.ActivityNotFoundException: Unable to find explicit activity class {cn.itcast.activitybasic/cn.itcast.activitybasic.SecondActivity};  
have you declared this activity in your AndroidManifest.xml?
```



在清单文件中引用Activity的方式

如果Activity所在的包与AndroidManifest.xml文件的<manifest></manifest>标签中通过package属性指定的包名一致，则android:name属性的值可以设置为“.Activity名称”，以SecondActivity为例，示例代码如下：

```
<activity  
    android:name=".SecondActivity">  
</activity>
```





4.2.3 启动和关闭Activity

1

启动Activity : startActivity()方法

以启动SecondActivity为例，示例代码如下

```
Intent intent = new Intent(MainActivity.this,SecondActivity.class);
startActivity(intent);
```

2

关闭Activity : finish()方法

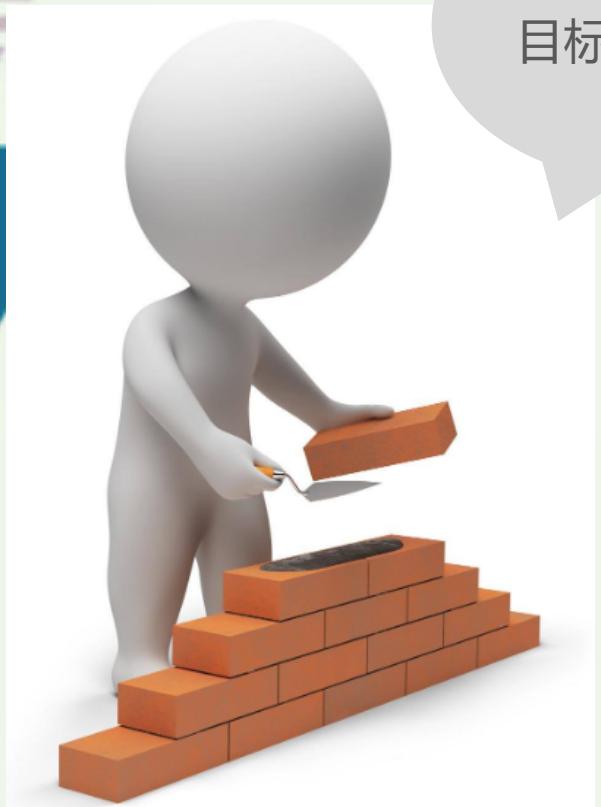
如果想要关闭当前的Activity，可以调用Activity提供的finish()方法。



4.3

Intent与IntentFilter

4.3 Intent与IntentFilter



- 掌握两种Intent类型的使用方式，能够灵活使用Intent的两种类型
- 掌握IntentFilter属性的匹配规则，能够独立配置IntentFilter的3个属性

4.3.1 Intent

Intent被称为意图，它不仅可以指定当前组件要执行的动作，还可以在不同组件之间进行数据传递。根据开启目标组件的方式不同，Intent被分为两种类型，分别为显式Intent和隐式Intent。

显式意图可以直接通过名称开启指定的目标组件

显式
意图

隐式意图通过指定action和category等属性，系统根据这些信息进行分析后寻找目标Activity

隐式
意图



4.3.1 Intent

显式意图

显式Intent指的是直接指定目标组件，例如，使用Intent显式指定要跳转的目标Activity，示例代码如下：

```
Intent intent = new Intent(this, SecondActivity.class);
```

```
startActivity(intent);
```

启动Activity

当前Activity

要启动的Activity



4.3.1 Intent

隐式意图

隐式Intent不会明确指出需要激活的目标组件，它被广泛地应用在不同应用程序之间传递消息。

Android系统会使用IntentFilter匹配属性action、data、category，这3个属性的具体介绍如下：

- **action**：表示Intent对象要完成的动作。
- **data**：表示Intent对象中传递的数据。
- **category**：表示为action添加的额外信息。



// Action

- action是一个字符串，action的匹配规则是Intent中的action必须能够和过滤规则中的action匹配，这里说的匹配是指action的字符串值完全一样。
- 一个过滤规则中可以有多个action，那么只要Intent中的action能够和过滤规则中的**任何一个**action相同即可匹配成功。

// Category

- category也是一个字符串，系统预定义了一些字符串，同时我们也可以定义自己的category。
- category的匹配规则和action不同，如果Intent中含有category，不管有几个，它都必须是匹配规则中定义过的。（这是它和action最大的区别，**每一个**必须是匹配规则也就是IntentFilter中定义过的category）
- 如果Intent中没有匹配category，系统默认添加category “android.intent.category.DEFAULT”



// data

- data的匹配规则和action类似，如果IntentFilter中定义了data，那么Intent中也要有定义了data。
- data的语法结构如下：

```
<data android:scheme="string"  
      android:host="string"  
      android:port="string"  
      android:path="string"  
      android:pathPattern="string"  
      android:pathPrefix="string"  
      android:mimeType="string"/>
```

https://blog.csdn.net/weixin_38664232/article/details/84333320



4.3.1 Intent

隐式意图

在清单文件中，配置SecondActivity的action为“cn.itcast.START_ACTIVITY”的代码如下所示：

```
<activity android:name=".SecondActivity">
    <intent-filter>
        <action android:name="cn.itcast.START_ACTIVITY"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

设置action动作，当代码中的action与该action相匹配时启动该组件。



4.3.1 Intent

隐式意图

在程序的MainActivity中开启SecondActivity的示例代码如下：

```
Intent intent = new Intent();
intent.setAction("cn.itcast.START_ACTIVITY");
startActivity(intent);
```

设置action动作，当与清单文件中的action相匹配时启动目标组件。

注意：在使用隐式Intent开启Activity时，系统会默认为该Intent添加category的属性name的值为“android.intent.category.DEFAULT”，所以将SecondActivity对应的<category>标签中，属性android:name的值设置为“android.intent.category.DEFAULT”。



4.3.2 IntentFilter

当发送一个隐式Intent后，Android系统会将它与程序中的每一个组件的过滤器进行匹配，匹配属性有action、data和category，需要这3个属性都匹配成功才能唤起相应的组件。

(1) action属性匹配规则

action属性用来指定Intent对象的动作，具体示例代码如下：

```
<intent-filter>  
    <action android:name="android.intent.action.EDIT" />  
    <action android:name="android.intent.action.VIEW" />  
    .....  
</intent-filter>
```

- 只要Intent携带的action与其中一个<intent-filter>标签中action的声明相同，action属性就匹配成功。

注意：在清单文件中为Activity添加<intent-filter>标签时，必须添加action属性，否则隐式Intent无法开启该Activity。



4.3.2 IntentFilter

(2) data属性匹配规则

data属性用来指定数据的URI或者数据MIME类型，它的值通常与Intent的action属性有关联，具体示例代码如下：

```
<intent-filter>
    <data android:mimeType="video/mpeg" android:scheme="http....." />
    <data android:mimeType="audio/mpeg" android:scheme="http....." />
    .....
</intent-filter>
```

- 隐式Intent携带的数据只要与IntentFilter中的任意一个data声明相同，data属性就匹配成功。



>>> 4.3.2 IntentFilter

(3) category属性匹配规则

category属性用于为action添加额外信息，一个IntentFilter可以不声明category属性，也可以声明多个category属性，具体示例代码如下：

```
<intent-filter>
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    .....
</intent-filter>
```

- 一个IntentFilter可以不声明category属性，也可以声明多个category属性。
- 隐式Intent中声明的category必须全部能够与某一个IntentFilter中的category属性匹配才算匹配成功。

注意：IntentFilter中罗列的category属性数量必须大于或者等于隐式Intent携带的category属性数量时，category属性才能匹配成功。如果一个隐式Intent没有设置category属性，那么他可以通过任何一个IntentFilter（过滤器）的category匹配。

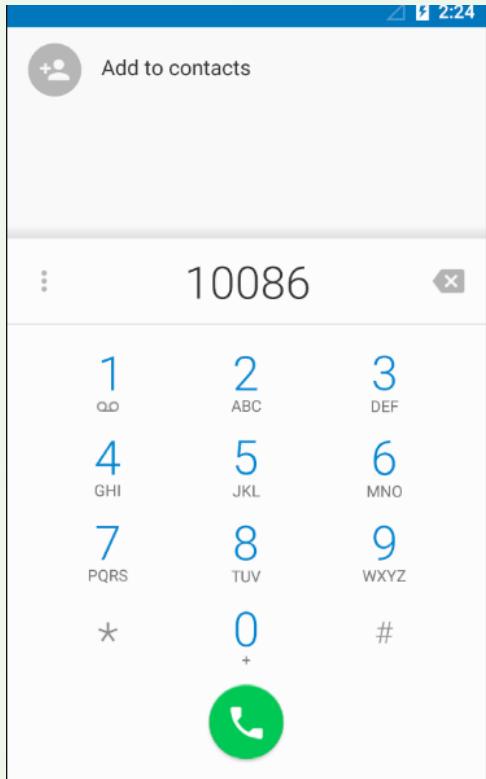


标准Activity action

常量	说明
ACTION_MAIN	作为初始的Activity启动，没有数据输入输出
ACTION_VIEW	将数据显示给用户
ACTION_ATTACH_DATA	用于指示一些数据应该附属于其他地方
ACTION_EDIT	将数据显示给用户用于编辑
ACTION_PICK	从数据中选择一项，并返回该项
ACTION_CHOOSER	显示Activity选择器，允许用户在继续前按需选择
ACTION_GET_CONTENT	允许用户选择特定类型的数据并将其返回
ACTION_DIAL	使用提供的数字拨打电话
ACTION_CALL	使用提供的数据给某人拨打电话
ACTION_SEND	向某人发送消息，接收者未指定
ACTION_SENDTO	向某人发送消息，接收者已指定
ACTION_ANSWER	接听电话
ACTION_INSERT	在给定容器中插入空白项
ACTION_DELETE	从容器中删除给定数据
ACTION_RUN	无条件运行数据
ACTION_SYNC	执行数据同步
ACTION_PICK_ACTIVITY	挑选给定Intent的Activity，返回选择的类
ACTION_SEARCH	执行查询
ACTION_WEB_SEARCH	执行联机查询
ACTION_FACTORY_TEST	工厂测试的主入口点

// 动手练一练

通过Intent,实现拨打电话的操作。



```
public void onClick(View v) {  
    Intent intent = new Intent(Intent.ACTION_DIAL);  
    intent.setData(Uri.parse("tel:10086"));  
    startActivity(intent);
```



4.4

Activity之间的跳转

4.4 Activity之间的跳转



- 掌握Activity之间的数据传递方式，能够使用Intent类与Bundle类传递数据
- 掌握Activity之间的数据回传方式，能够完成Activity之间的数据回传



4.4.1 Activity之间的数据传递

Android提供的Intent可以在界面跳转时传递数据。使用Intent传递数据有两种方式。

1

使用Intent的putExtra()方法传递数据

2

使用Bundle类传递数据



>>> 4.4.1 在Activity之间的数据传递

使用Intent的putExtra()方法传递数据

Activity之间需要传递不同类型的数据，所以Android系统提供了多个重载的putExtra()方法。

m void putExtra (String name, boolean value)	Intent
m void putExtra (String name, boolean[] value)	Intent
m void putExtra (String name, Bundle value)	Intent
m void putExtra (String name, byte value)	Intent
m void putExtra (String name, byte[] value)	Intent
m void putExtra (String name, char value)	Intent
m void putExtra (String name, CharSequence value)	Intent
m void putExtra (String name, double value)	Intent
m void putExtra (String name, double[] value)	Intent
m void putExtra (String name, float value)	Intent
m void putExtra (String name, float[] value)	Intent
m void putExtra (String name, int value)	Intent
m void putExtra (String name, int[] value)	Intent
m void putExtra (String name, long value)	Intent
m void putExtra (String name, long[] value)	Intent
m void putExtra (String name, Parcelable value)	Intent
m void putExtra (String name, Parcelable[] value)	Intent
m void putExtra (String name, Serializable value)	Intent
m void putExtra (String name, short value)	Intent
m void putExtra (String name, short[] value)	Intent
m void putExtra (String name, String value)	Intent
m void putExtra (String name, String[] value)	Intent



>>> 4.4.1 在Activity之间的数据传递

使用Intent的putExtra()方法传递数据

```
Intent intent = new Intent();
intent.setClass(MainActivity.this,SecondActivity.class);
intent.putExtra("studentName","王晓明");
intent.putExtra("englishScore",98);
intent.putExtra("isPassed",true);
startActivity(intent);
```

在MainActivity中将数据
传递给SecondActivity

```
Intent intent = getIntent();
String name = intent.getStringExtra("studentName");
int englishScore = intent.getIntExtra("englishScore",0);
boolean isPassed = intent.getBooleanExtra("isPassed",true);
```

在SecondActivity 中获取
MainActivity传递来的数据



>>> 4.4.1 在Activity之间的数据传递

使用Bundle类传递数据

```
Intent intent = new Intent();
intent.setClass(this,SecondActivity.class);
Bundle bundle = new Bundle();
bundle.putString("account", "王小明");
intent.putExtras(bundle);
startActivity(intent);
```

将用户名数据封装到Bundle对象中

```
Bundle bundle = getIntent().getExtras();
String account = bundle.getString("account");
```

通过Bundle对象获取用户名信息



// Intent传递数据（1）

使用Bundle在Activity之间交换数据

- Bundle是一个字符串值到各种Parcelable类型的映射，用于保存要携带的数据包。
- Bundle是一个利用key-value（键-值）对保存数据的工具。我们可以根据其中的key来获取具体的内容（value）。

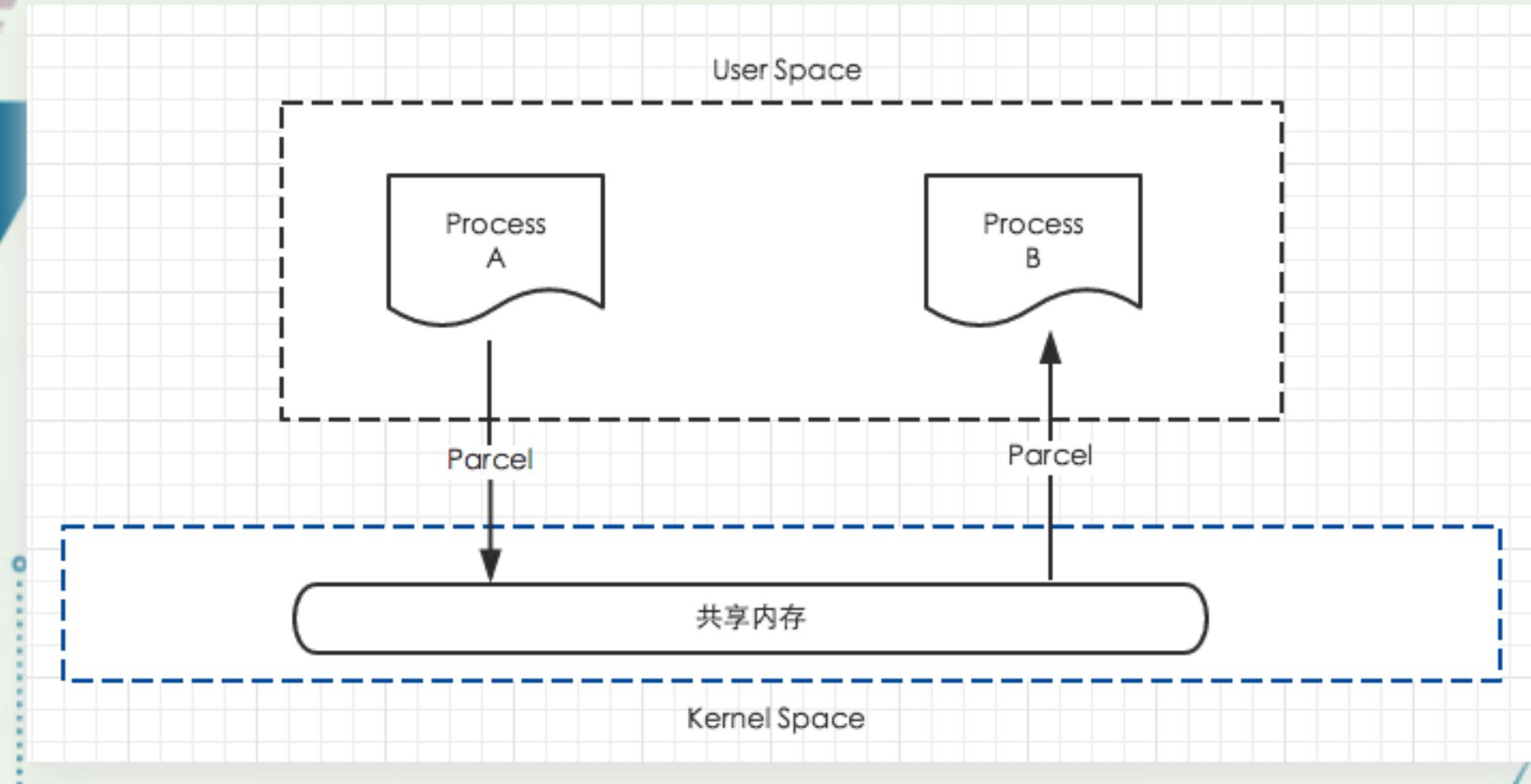


Parcelable接口 和序列化

- Parcelable不得不先提一下Serializable接口。
- Serializable是Java为我们提供的一个标准化的序列化接口,那什么是序列化呢?
- 进行Android开发的时候, 无法将对象的引用传给Activities, 我们需要将这些对象放到一个Intent或者Bundle里面, 然后再传递。简单来说就是将对象转换为可以传输的二进制流(二进制序列)的过程,这样我们就可以通过序列化,转化为可以在网络传输或者保存到本地的流(序列),从而进行传输数据,那反序列化就是从二进制流(序列)转化为对象的过程.
- Parcelable是Android为我们提供的序列化的接口, Parcelable相对于Serializable的使用相对复杂一些,但Parcelable的效率相对Serializable也高很多。

Parcel翻译过来是打包的意思

Parcel可以包含原始数据类型（用各种对应的方法写入，比如writeInt(), writeFloat()等），可以包含Parcelable对象，





// 发送数据的Activity

- 将数据存放在Bundle对象中，并将其添加到Intent对象中，可以通过下面的代码实现。

```
Bundle bundle=new Bundle();          //创建并实例化一个Bundle对象  
bundle.putCharSequence("user", user); //保存用户名  
bundle.putCharSequence("pwd", pwd);   //保存密码  
intent.putExtras(bundle);           //将Bundle对象添加到Intent对象中
```



接收数据的Activity

- 获取传递过来的字符串类型的用户名和密码，可以使用下面的代码

```
Intent intent=getIntent();          //获取Intent对象  
Bundle bundle=intent.getExtras();   //获取传递的数据包  
bundle.getString("user");           //获取输入的用户名  
bundle.getString("pwd");            //获取输入的密码
```