Android App的开发
四大组件

本章大纲

Activity （活动）

Service（服务）

BroadcastReceiver（广播接收器）

Content Provider（内容提供者）

Activity概念

- Activity是Android应用中最重要的核心组件，每一个应用屏幕就是一个Activity；这意味着，要创建多屏幕的应用，必须创建多个Activity。

Activity

Activity是会显示视图控制组件的用户接口，并对事件作出响应， Activity是Android应用程序的最基本的组件。

- Android应用程序中一个单独的屏幕通常就是一个Activity。它上面可以显示一些控件，也可以监听处理用户的事件并做出响应。
- 每个屏幕通常都被实现为一个独立的Activity类，即继承自AppCompatActivity基类。
- 它是整个应用程序的门面，主要负责数据的显示与交互

Activity概念

- Android中的Activity的使用基本上分为以下三大类问题：
  - 如何创建多屏幕（如何创建多个Activity ）？
  - 屏幕与屏幕之间如何切换（Activity之间的跳转）？
  - 屏幕是何时产生何时消亡的（Activity的生命周期）？

创建新的Activity

- 创建新的Activity的基本流程是：
  - 创建新的类直接或间接继承Activity类（src/指定包/目录下）
  - 为该Activity类绑定布局（res/layout/目录下）
  - 在AndroidManifest.xml文件中注册该Activity

Activity跳转简介

一个Android应用中包含多个Activity，Activity之间必然存在某种跳转关系。

Activity跳转的基本原理

- 在Android中，Activity与Activity之间的跳转是借助Intent对象来实现的。
  - Intent对象用来在Activity与Activity之间传递请求消息和响应消息。
  - 也就是说，Intent对象充当了HTTP协议中的请求对象和响应对象双重作用。
  - Android中的三大核心组件，活动(Activity)、服务(Service)和广播接收器(BroadcastReceiver)，都是通过Intent来启动或激活的。

Activity跳转

- 发送请求的Activity页面
  1. 创建Intent对象：
     - Intent i = new Intent( );
  2. 设置请求目的地：
     - i.setClass(上下文, 待启动的Activity.class);
     - i.setAction(目的Activity字符串);
  3. 携带数据（可选）
  4. 发送请求（启动新的Activity）：
     - startActivity(Intent对象);
     - startActivityForResult(Intent对象, 请求码);

Activity跳转（携带数据）

- 发送请求的Activity页面

  3. 携带数据（可选）：

     - 直接添加基本类型参数：`i.putExtra( key, value );`
     - 传递类的对象（需要序列化对象后才可传递）

Activity跳转（携带数据）

- 发送请求的Activity页面

  3. 携带数据（可选）：

     – 创建复杂数据对象：借助Bundle对象实现

       » 创建Bundle对象：`Bundle b = new Bundle( );`

       » 为Bundle对象添加数据：`b.putString( );`、`b.putSerializable( );`、……

       » 把Bundle对象添加到Intent对象中：`i.putExtra(Bundle对象);`

http://developer.android.com/reference/android/os/Bundle.html

Activity跳转（被请求页面处理请求）

- 被请求的Activity页面
  - 获得Intent对象（请求对象）：
    - `Intent request = getIntent( );`
  - 获得请求参数：
    - `request.getIntExtra( );`：返回基本int类型数据
    - `request.getExtras( );`：返回Bundle对象

    http://developer.android.com/reference/android/content/Intent.html

Activity跳转（无响应）

- 完成Activity的跳转的步骤：
  - 注册跳转的触发事件；
  - 构造跳转Intent，加入参数；
  - 进行activity的跳转；
  - 在跳转到的activity中接受传入的参数。

Activity跳转（被请求页面设置响应）

- 被请求的Activity页面
  - 获得Intent对象（响应对象）：
    - Intent response = new Intent( );
  - 添加响应消息：
    - response.putIntExtra( );：添加基本int类型数据
    - response.putExtras( );：添加Bundle对象
  - 实现响应：
    - this.setResult(int resoponseCode, Intent 响应对象)
    - finish()

Activity跳转（请求页获取响应消息）

- 请求的Activity页面

```
// 请求页面接收响应并处理响应消息
protected void onActivityResult(
        int requestCode,     //请求码
        int resultCode,      //响应码
        Intent data          //Intent响应对象
        ) {
    //接收响应消息
    int i = data.getIntExtra("age", 0);
    Bundle b = data.getExtras();
    //处理响应消息
}
```

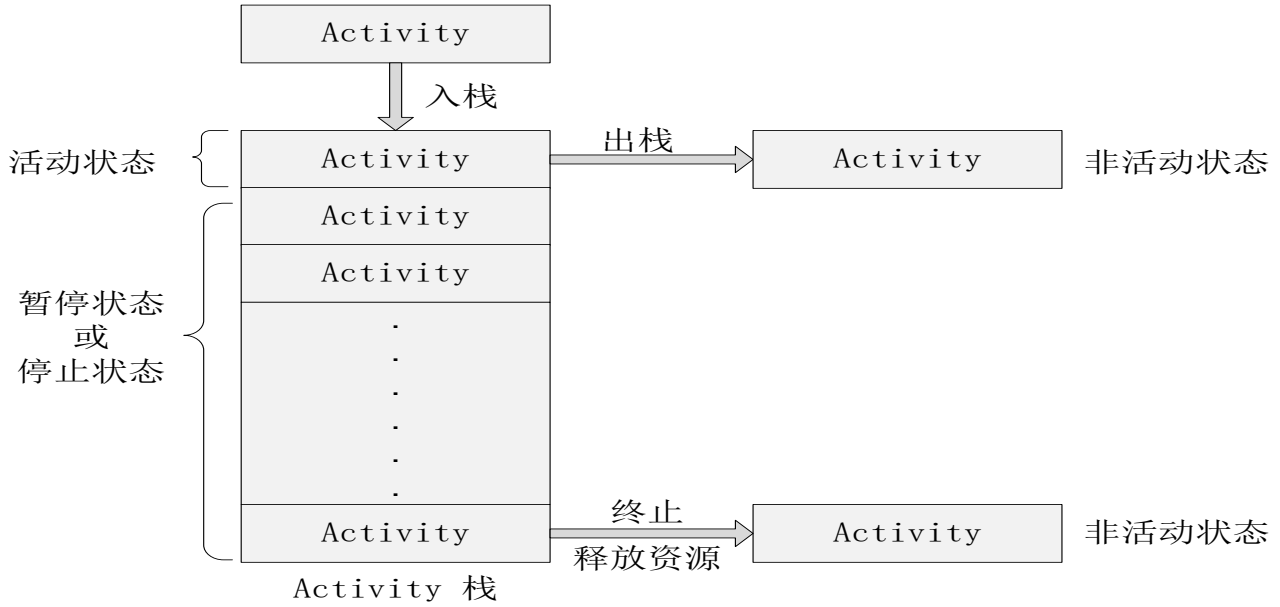
综合应用

父Acvity：EarthActivity，去火星（带参数）

子Acvity：MoonActivity，MarsActivity（返回地球，并带响应信息）

Activity活动栈

- Android应用可能含有多个Activity，管理这些Activity之间的先后次序关系，需要**Activity活动栈机制**



Activity的活动状态

- 随着Activity的创建、销毁，Activity在内存中有4种状态表现形式：
  - 活动状态：当前Activity在Activity活动栈中处于最上层，完全能被用户看到，并能够与用户进行交互。
    - 正在运行的屏幕即为此种状态。

Activity的活动状态

- 暂停状态：当前Activity在界面上被部分遮挡，不再处于用户界面的最上层，不能够与用户进行交互。
  - 若启动一个新的Activity（以对话框形式展示），则原来的Activity就处于暂停状态。
  - 处于暂停状态的Activity仍然保留用户的状态信息，但在系统内存不足时，可能会被系统杀死。

Activity的活动状态

- 停止状态：Activity在界面上完全不能被用户看到，也就是说这个Activity被其他Activity全部遮挡。
  - 例如：在Activity中，用户按下“Home”键时，原来的Activity就处于停止状态。
  - 处于停止状态的Activity，仍然保留用户状态信息，但当系统内存不足时，会优先杀死该类Activity。

Activity的活动状态

- 非活动状态：不在以上三种状态中的Activity，处于非活动状态。
  - 被销毁的Activity即处于该类状态。

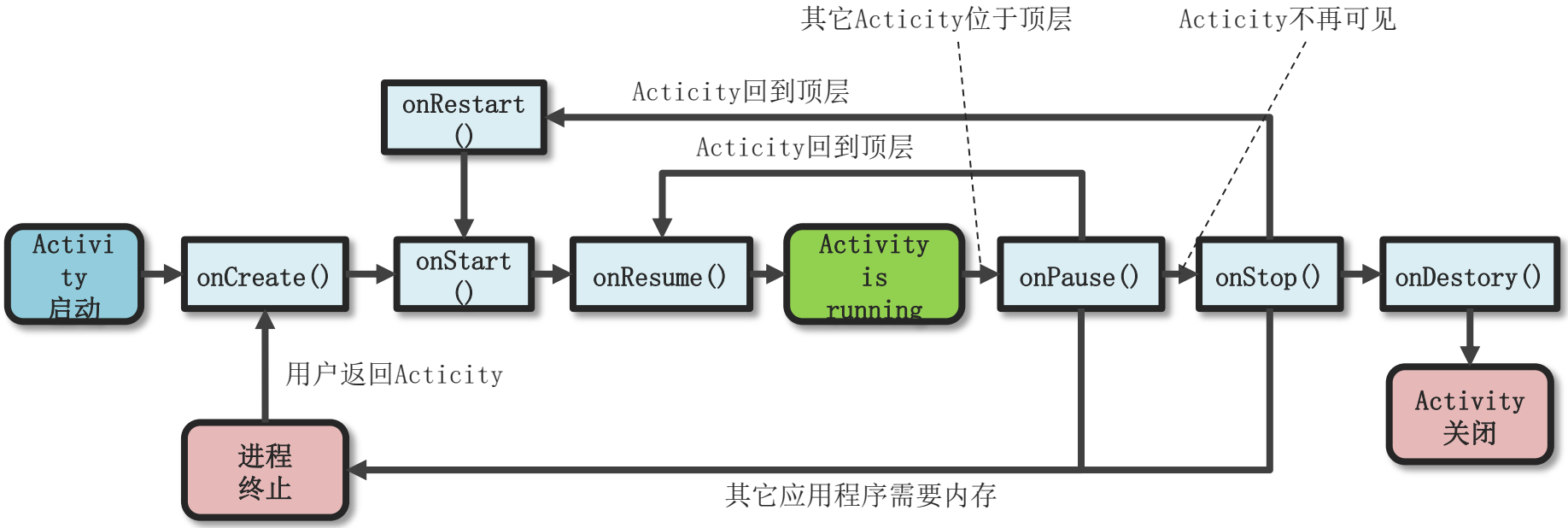
Activity活动状态之间的切换

- 当Activity各个活动状态之间发生切换时，会触发以下Activity回调方法：

回调方法	描述
onCreate(Bundle savedStatus)	创建 Activity 时被回调；进行 Activity 的初始化，例如创建 View、绑定数据或恢复状态信息等。
onStart()	启动 Activity 时（Activity 显示在屏幕上时）被回调。
onRestart()	当 Activity 从停止状态进入活动状态时被回调；此时可能需要恢复用户保存的数据。
onResume()	当 Activity 能够与用户交互时（或 Activity 从暂停状态恢复时）被回调；此时当前屏幕所对应的 Activity 处于 Activity 活动栈的栈顶。
onPause()	当 Activity 进入暂停状态时被回调；此时常需要保存持久性数据或释放占用的资源。
onStop()	当 Activity 进入停止状态时被回调。
onDestroy()	在 Activity 被终止前（即进入非活动状态前）被回调。

Activity活动状态之间的切换

- 当Activity各个活动状态之间发生切换时，会触发以下Activity回调方法：



本章大纲

Activity （活动）

Service（服务）

BroadcastReceiver（广播接收器）

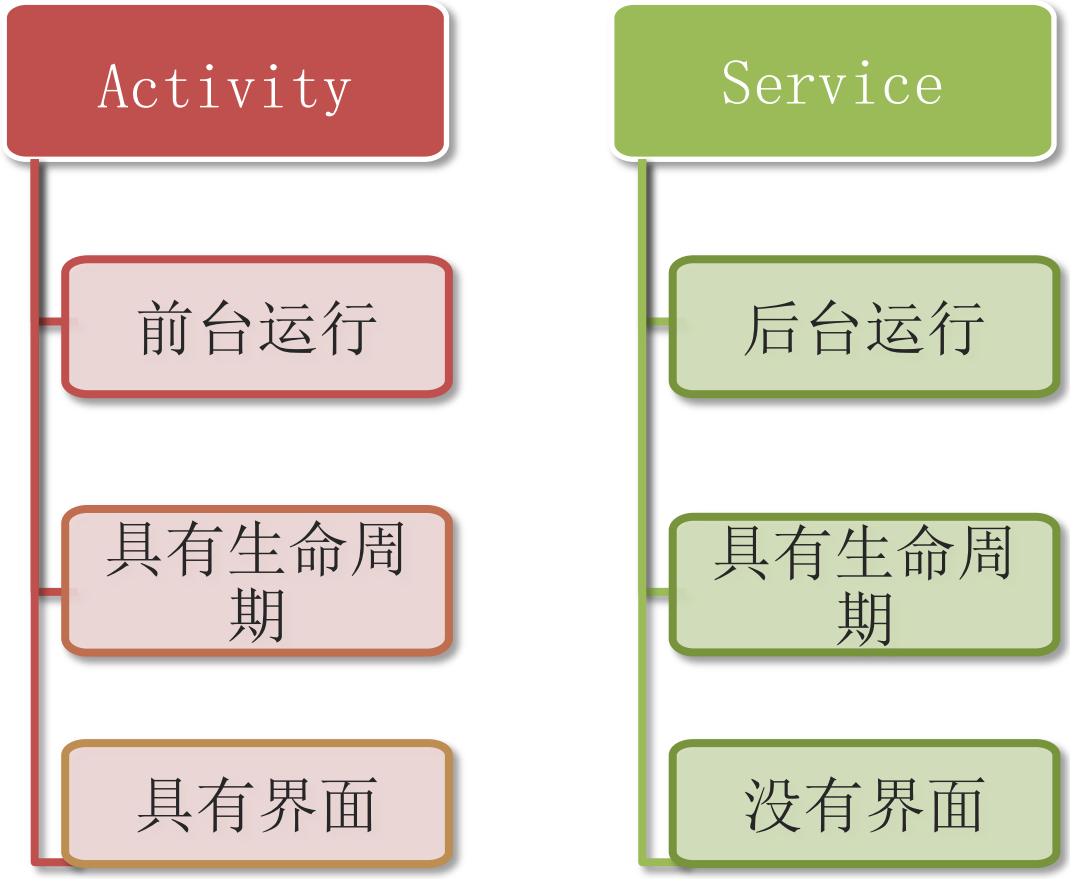
Content Provider（内容提供者）

Service 介绍

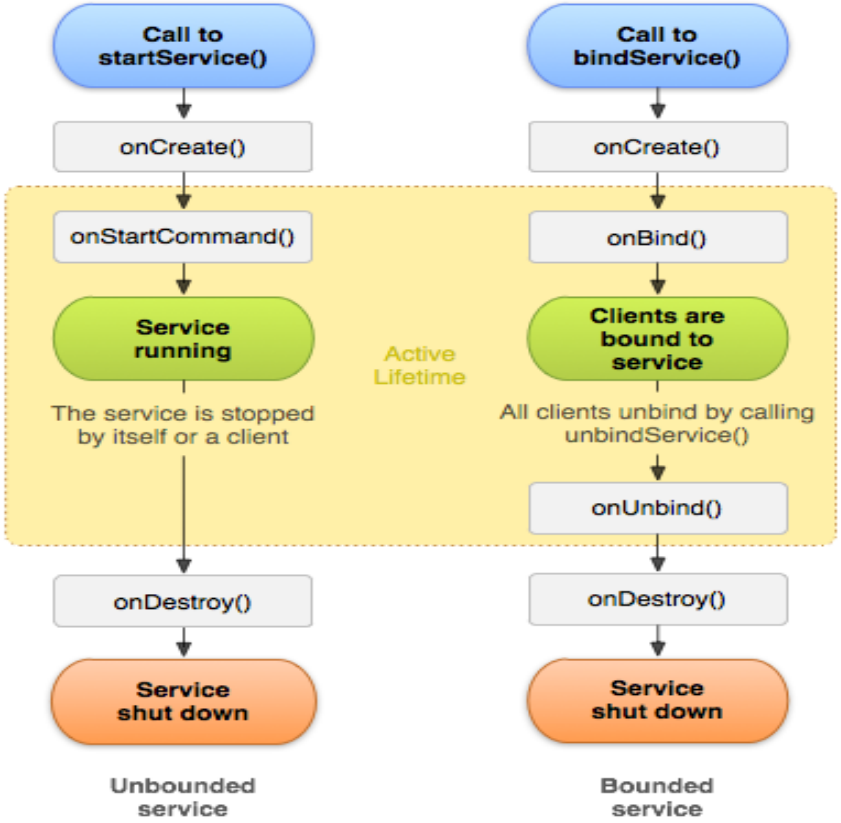
Service是具有一个较长生命周期且没有用户界面的程序，只能在后台运行，可以和其他组件进行交互。

例如：一个音乐播放器。用户可以再设备上一边播放音乐一边进行别的操作。

Service VS Activity



Service 生命周期



Service使用

Android Service两种启动方式的区别

1、采用start的方式开启服务

特点：如果服务已经开启，不会重复的执行onCreate()， 而是会调用onStartCommand()。服务停止的时候调用 onDestory()。服务只会被停止一次。一旦服务开启跟调用者(开启者)就没有任何关系了。开启者退出了，开启者挂了，服务还在后台长期的运行。

开启者不能调用服务里面的方法。

2、采用bind的方式开启服务

bind的方式开启服务，绑定服务，调用者挂了，服务也会跟着挂掉。

绑定者可以调用服务里面的方法。

Service使用

1、直接创建Service，继承Service，创建成功后，会在AndroidManifest.xml进行注册

2、实现以下函数

void onCreate():第一次创建后回调

void onDestroy():关闭前回调

void onStartCommand(intent,flags,startID):实现核心业务

3、创建Activity调用启动/关闭Service

startService(intent);

stopService(intent);

Service实例

```
//启动Service
public void startClick(View v) {
    Intent i = new Intent(this, FirstService.class);
    startService(i);
}
//结束Service
public void stopClick(View v) {
    Intent i = new Intent(this, FirstService.class);
    stopService(i);
}
```

Service实例

```
public int onStartCommand(Intent intent, int flags, int startId) {
    for(int i=0;i<100;i++){
        System.out.println("***************onStartCommand"+i+"**"+Thread.currentThread().getName());
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        if(i==20){
            this.stopSelf();
            break;
        }
    }
    return super.onStartCommand(intent, flags, startId);
}
```

Service总结

1. 服务只被创建一次，可以通过外部调用stopService(intent)或stopSelf()
2. 当执行一个已启动的服务，或直接调用onStartCommand方法来执行业务
3. 默认情况下服务与主线程在同一个进程中的同一个线程中执行，如果服务执行一个比较耗时的操作，我们必须使用子线程来完成工作，避免阻塞主线程
4. 使用startService(intent);启动的服务，在没有关闭之前会一直在后台运行

本章大纲

Activity （活动）

Service（服务）

BroadcastReceiver（广播接收器）

Content Provider（内容提供者）

生活中的广播

广播机制的总结

- 广播发送者
  - 可发送一种或多种广播
  - 不关心是谁接收
  - 具有实时性
- 广播接收者
  - 可接收一种或多种广播
  - 不关心是谁发送
  - 具有实时性

BroadcastReceiver简介

BroadcastReceiver组件本质上是一种全局的监听器，因此它的主要作用是实现系统间不同组件之间的通讯。

Android中广播机制的介绍

- Android手机中有很多应用采用广播机制：
  - 电话的接听和拨打
  - 短信的接收和发送
  - 电池的状态
  - 系统的闹钟
  - 手机连接电脑
  - 手机脱离电脑
  - ……

BroadcastReceiver的创建步骤

1. 在Activity事件中构建Intent，使用sendBroadcast方法发送广播
2. 定义一个BroadcastReceiver，覆盖onReceive()方法来响应事件（new/Other/BroadcastReceiver）
3. 注册BroadcastReceiver（在代码中或者AndroidManifest.xml文件中 ）

BroadcastReceiver实例

```
public void sendBroadcast(View v){
    Intent i =new Intent("com.edu2act.MY_BROAD");
    i.putExtra("uname","tom");
    this.sendBroadcast(i);
}
```

```
<receiver
    android:name=".broadcastDemo.MyReceiver1"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="com.edu2act.MY_BROAD" ></action>
    </intent-filter>
</receiver>
```

```
@Override
public void onReceive(Context context, Intent intent) {
    String uname = intent.getStringExtra("uname");
    Toast.makeText(context,uname,Toast.LENGTH_LONG).show();
}
```

BroadcastReceiver生命周期

- BroadcastReceiver生命周期只有十秒左右，如果在onReceive内做超过10秒内的事情，就会报ANR（无响应）的错误信息。
- onReceive方法中不能加入比较耗时的操作，否则系统会认为程序无响应，如果一定要执行耗时的操作的话，一般通过Intent启动一个Service来完成。

本章大纲

Activity （活动）

Service（服务）

BroadcastReceiver（广播接收器）

Content Provider（内容提供者）

数据共享

Android中存在多个应用程序，应用程序间的数据如何共享呢？

联系人数据

ContentProvider介绍

- 为了在**应用程序之间共享数据**，Android提供了ContentProvider，这是一种不同应用之间共享数据的标准API：
  - 当应用希望提供数据时，就提供ContentProvider
  - 其他应用通过ContentResolver来操作
- 注意：
  - ContentProvider需要在AndroidManifest.xml中注册
  - 一旦应用提供CP，不论应用启动与否，都可被操作

ContentProvider实例

添加权限：

```
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

```
public void viewClick(View v) {
    ContentResolver contentResolver = this.getContentResolver();
    String id = null;
    String mimetype = null;
    Cursor cursor = contentResolver.query(android.provider.ContactsContract.Contacts.CONTENT_URI,
            new String[]{ContactsContract.Contacts._ID}, null, null, null);
    while (cursor.moveToNext()) {
        id = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts._ID));
        Cursor contactInfoCursor = contentResolver.query(android.provider.ContactsContract.Data.CONTENT_URI,
                new String[]{ContactsContract.Data.CONTACT_ID,
                        ContactsContract.Data.MIMETYPE,
                        ContactsContract.Data.DATA1,
                        ContactsContract.Data.DATA15},
                android.provider.ContactsContract.Data.CONTACT_ID + "=" + id, null, null);
        while (contactInfoCursor.moveToNext()) {
            mimetype = contactInfoCursor.getString(contactInfoCursor.getColumnIndex(ContactsContract.Data.MIMETYPE));
            String value = contactInfoCursor.getString(contactInfoCursor.getColumnIndex(ContactsContract.Data.DATA1));
            if (mimetype.contains("/name")) {

                System.out.println("姓名是" + value);

            } else if (mimetype.contains("/im")) {
                System.out.println("QQ是" + value);
            } else if (mimetype.contains("/email")) {
                System.out.println("邮箱是" + value);
```