

概述

- 6.14
 1. 完成开发环境配置，工具安装和使用
 2. 疯狂Android讲义（1页-46页）
 1. Android是什么
 2. Android是干什么的
 3. Android默认使用gradle作为项目管理工具，需掌握Gradle
 4. 学习IDE的使用，Android Studio，Android SDK，以及Android SDK提供的模拟器和调试器
 5. 如何创建Android应用
 6. Android应用的目录结构，初步了解其作用
- 6.15
 1. 进一步了解Android应用的目录结构
 2. 初步了解Android应用的四大基本组件：Activity、Service、BroadcastReceiver和ContentProvider
 3. 初步了解应用内不同组件的通信载体Intent以及IntentFilter
 4. 学习所有UI组件的父类view组件
 5. 学习布局管理器UI组件
 6. 学习TextViewUI组件

配置已完成：jdk，AndroidStudio，Git，邮箱

- jdk问题
 - 安装jdk1.8出现内部错误61003
 - 原因：电脑没有安装visual c++2015 redistributable
 - 解决办法：安装visual c++2015之后再次安装jdk安装正常。
- AndroidStudio问题
 - 安装AndroidSDK失败
 - 原因：需要访问google网络
 - 解决办法：
 1. 下载v2rayN代理工具和自身微服务器上的v2ray代理服务访问。
 2. 在打开v2rayN时，提示需要.net framework 4.8，
 3. 安装.net framework 4.8时，提示“安装未成功，已处理证书链,但是在不受信任提供程序信任的根证书中终止”，原因是计算机中没有相应的受信任证书，需要导入微软的证书 MicrosoftRootCertificateAuthority2011.cer
 4. 导入证书之后，再次安装.net framework 4.8提示“时间戳签名和或证书无法验证或已损坏”，原因是系统版本太低，需要更新系统。
 5. 更新系统后成功安装.net framework 4.8,正常打开v2rayN，配置后，解决无法访问google的问题。
- git问题
 - git clone 远程仓库提示 Failed to connect to github.com port 443: Timed out
 - 原因：代理没有配置对

- 解决办法:

- ```
//输入如下两端命令
//server和端口根据自己的v2rayN配置修改
git config --global http.proxy http://127.0.0.1:10809
git config --global http.proxy socks5://127.0.0.1:10808
```

# 疯狂Android讲义

## 1.Android应用和开发环境

### Android的简介

- Android是由Andy Rubin创立的一个手机操作系统，后被Google收购。
- 目标：Google希望能与各方共同建立一个标准化、开放式的移动电话软件平台，从而在移动产业内形成一个开放式的操作平台。

### Android9.x平台特性

- Android的底层建立在linux之上，主要由操作系统、中间件、用户界面和应用软件4层组成
  - 设计思路：采用一种被称为软件叠层(Software Stack)的方式进行构建。
  - 作用
    - 使得层与层之间互相分离，明确各层的分工
    - 保证了层与层的低耦合，当下层的层内或层下发生改变时，上层应用程序无需任何改变
  - 和网络以及java web的后端一样都使用了分层的思想去解决问题。
- Android系统主要由6部分组成。**序号从小到大，越来越底层。**
  1. 系统App层
    - Android系统将包含一系列核心App，如电话拨号，联系人。
    - 普通开发者开发的各种Android程序，都属于这一层
  2. Java API框架层
    - 提供了大量API供开发者使用
    - 调用原生C/C++库的服务。
  3. 原生C/C++库
    - Android包含一套被不同组件所使用的C/C++库的集合。一般来说，Android应用开发者不能直接调用这套C/C++库集，而是通过它上面的JavaAPI框架调用这些库
    - SQLite：供所有应用程序使用的功能强大的轻量级关系数据库
    - WebKit
    - OpenMAX
    - Libc
    - Media Framework
    - SGL
    - OpenGL ES
  4. Android运行时
    - 由Android核心库和ART ( Android runtime ) 组成
  5. 硬件抽象层 ( HAL )
    - 主要提供了对linux内核的封装，向上提供蓝牙、摄像头等设备的编程接口，向下隐藏底层的实现细节。
  6. Linux内核

- 提供了安全性，内存管理、进程管理、网络协议栈和驱动模型等核心服务。

## Gradle自动化构建

---

- Gradle的运行需要Java\_Home环境变量
- 使用gradle 4.10.2

### Gradle文件结构

- bin：包含gradle的命令-gradle
- docs：包含用户手册、DSL参考指南、API文档
- lib：包含Gradle核心，以及它依赖的JAR包
- media：主要包含Gradle的一些图标
- sample：样例
- src：Gradle源代码

### Gradle项目结构

- Gradle构建文件的默认名字为build.gradle
- project:存放整个项目的全部资源
  - src:分类存放各种源文件、资源文件
    - main：主要存放与项目有关的源文件和资源
      - java：存放与项目相关的资源
      - resource：存放与项目相关的资源
    - test：主要存放与测试有关的源文件和资源
      - java：存放与测试相关的资源
      - resource：存放与测试相关的资源
  - build：存放编译后的class文件、资源的文件夹，该文件夹与src具有对应关系
  - libs：存放第三方JAR包的文件夹
  - build.gradle：gradle构建文件

### 构建文件

- Gradle采用**领域对象模型**的概念来组织构建文件，在整个构建文件中涉及最核心的API
- Project：代表项目，通常一份构建文件代表一个项目。Project包含大量属性和方法
- TaskContainer：任务容器。每个Project都会维护一个TaskContainer类型的tasks属性。Project和TaskContainer由一一对应的关系
- Task：代表Gradle要执行的一个任务。Task允许指定它依赖的任务、该任务的类型，也可以通过configure（）方法配置任务。它还提供doFirst（）、doLast（）方法添加Action。
  - Action对象和Closure对象都可以代表Action。
  - Closure代表一个闭包，所以Action实际上就代表一个代码块

### Task创建

- 调用Project的task（）方法创建Task
- 调用TaskContainer的create（）方法创建Task

## Task指定如下3个常用属性

- dependsOn：指定该Task所依赖的其他Task
- type:指定该task的类型
- 代码块：通过传入的代码块参数配置Task

```
◦ task hello{
 println "hello world"
}
//使用
gradle hello
```

## 属性定义

### 1. 为已有的属性指定属性值

- Project、Task都是Gradle提供的API，他们本身具有内置属性，因此可以在构建文件中为这些属性指定属性值
- project的常用属性
  - name：项目的名字
  - path：项目的绝对路径
  - description：项目的描述信息
  - buildDir：项目的构建结果存放路径
  - version：项目的版本号

```
■ version = 1.0
description = "project的属性"
task hello{
 description = "task的属性"
 println "hello world"
}
```

### 2. 通过ext添加属性

- Gradle中所有实现了ExtensionAware接口的API都可通过ext来添加属性

```
◦ ext.prop1 = "111"
ext{
 prop2 = "222"
}
task hello{
 prop3 = "333"
 println "hello world"
}
```

### 3. 通过-P选项添加属性

### 4. 通过JVM参数添加属性

## 增量式构建

- 背景：如果每次执行任务都没有造成任何改变，那么重复执行就没有意义
- 如何判断是否产生变化？
  - Gradle将每个任务当做一个黑盒子：只要该任务的输入部分和输出部分都没有改变，Gradle就认为该任务的执行没有造成任何改变
    - 输入：Gradle的task使用inputs属性来代表任务的输入
      - 是一个TaskInput的对象
      - 可以设置文件、文件集、目录、属性等
    - 输出：Gradle的task使用outputs属性来代表任务的输出
      - 该属性是一个TaskOutputs类型的对象
      - 可以设置文件、文件集、目录、属性等

```
task hello{
 //def 定义属性的关键字
 def sourceTxt = fileTree("source")
 def dest = file('dist.txt')
 inputs.dir sourceTxt
 outputs.file dest
 println "hello world"
}
```

## Gradle插件

- 应用插件就相当于引入了该插件包含的所有任务类型、任务和属性等，使得Gradle可执行插件中预先定好的任务

- ```
//使用java插件  
apply plugin: 'java'  
//查看构建文件支持的所有任务  
gradle tasks --all
```

java插件

- 任务
 - assemble:装配整个项目
 - build：装配并测试该项目
 - buildDependents：装配并测试该项目，以及该项目的依赖项目
 - classes：装配该项目的所有类
 - clean：删除构建目录的所有内容。属于Delete类型的任务
 - jar：将项目的所有类打包成JAR包
 - init:执行构建的初始化操作
 - wrapper:生成Gradle包装文件
 - compileJava：编译项目的所有主java源文件
 - processResources：处理该项目的所有主资源
- java插件要求将项目源代码和资源都放在src目录下，将构建生成的字节码文件和资源放在build目录下，插件会自动管理该目录下的两类源代码和资源
 - main：项目的主代码和资源
 - test：项目的测试代码和资源
- 可以在构建文件中通过sourceSets方法改变项目代码、资源的存储路径，第三方或额外依赖的源代码也是通过sourceSets方法配置

```
○ sourceSets{  
    testlib  
}
```

- 可以将testlib的java源代码放在src/testlib/java目录下，资源放在src/testlib/resources目录下

依赖管理

- 背景：在构建java项目时通常都要依赖一个或多个框架
- 步骤

1. 为Gradle配置仓库。配置仓库的目的是告诉Gradle到哪里下载JAR包

- ```
//定义仓库，在构建文件中
repositories{
 //使用Maven默认的中央仓库
 mavenCentral()
}
```

2. 为不同的组配置依赖JAR包。配置依赖JAR包的目的是告诉Gradle要下载哪些JAR包

- 组：相当于一个文件夹，里面放不同的jar，谁要用就直接复制这个文件夹，而不需要一个一个jar的去复制

- ```
configurations{  
    group1  
}
```

- 使用dependencies来为依赖组配置JAR包

- group：JAR所属的组织名
- name：JAR名称
- version：JAR版本

- ```
dependencies{
 group1 group:'commons-logging',name:'commons-
logging',version:'1.2'
 //或者
 group1 'commons-logging:commons-logging:1.2'
}
```

- java插件提供的组

- implementation:主项目源代码(src/main/java)所依赖的组。
- compileOnly:主项目源代码(src/main/java)编译时才依赖的组。
- runtimeOnly:主项目源代码(src/main/java)运行时才依赖的组。
- testImplementation:项目测试代码(src/test/java)所依赖的组。
- testCompileOnly:项目测试代码(src/test/java)编译时才依赖的组。
- testRuntimeOnly:项目测试代码(src/test/java)运行时才依赖的组。
- archives:项目打包时依赖的组。

## 自定义任务

- 自定义任务就是一个实现Task接口的Groovy类,该接口定义了大量抽象方法需要实现,因此一般自定义任务类会继承DefaultTask基类。自定义任务的Groovy类可以自定义多个方法,这些方法可作为Action使用。
- 使用了@TaskAction修饰的方法将会自动添加为该任务的,反之需要使用doLast或doFirst方法手动将它添加成Action

## Android环境搭建

### 1. 安装Android Studio

### 2. 安装Android SDK

#### ◦ 目录结构

- build-tools:存放不同版本的 Android项目编译工具。
- docs:存放Android SDK开发文件和API文档等。
- emulator:存放Android自带的各种模拟器程序。
- extras:存放Google提供的USB驱动、Intel提供的硬件加速等附加工具包。
- licenses:存放 Android的相关授权文档。
- patcher:存放Android 的一些兼容性补丁。
- platforms:存放不同版本的Android系统。刚解压缩时该目录为空。
- platform-tools:存放 Android平台相关工具。
- skins:存放针对不同Android模拟器的皮肤。
- sources:存放不同版本的Android系统的源代码。
- system-images:存放不同Android平台针对不同CPU架构提供的系统镜像。这些系统镜像用于启动、运行 Android模拟器。
- tools:存放大量的Android开发、调试工具。

## 第一个Android应用

- Android应用程序建立在应用程序框架之上,所以Android编程就是面向应用程序框架 API编程——这种开发方式与编写普通的 Java或Kotlin应用程序并没有太大的区别,只是Android新增了一些API而已。
- 步骤
  1. 建立一个Android项目或Android模块
  2. 在XML布局文件中定义应用程序的用户界面
    - Android 把用户界面放在XML文档中定义,就可以让XML文档专门负责用户UI设置,而Java程序则专门负责业务实现这样可以**降低程序的耦合性**
    - res/layout存放Android的应用用户界面,类似于html
      - RelativeLayout:它代表一个相对布局
      - UI控件
        - Button:代表一个普通按钮。
        - TextView:代表一个文本框。
      - `android:id`:指定该控件的唯一标识,在Java或Kotlin程序中可通过 `findViewById("id")`来获取指定的 Android界面组件。
      - `android:layout_width`:指定该界面组件的宽度。如果该属性值为 `match parent`,则说明该组件与其父容器具有相同的宽度;如果该属性值为 `wrap content`,则说明该组件的宽度取决于它的内容——能包裹它的内容即可。

- `android:layout height`:指定该界面组件的高度。如果该属性值为 `match parent`，则说明该组件与其父容器具有相同的高度;如果该属性值为 `wrap content`，则说明该组件的高度取决于它的内容——能包裹它的内容即可。

### 3. 在Java或Kotlin代码中编写业务实现

- `main/java`存放Android的业务逻辑

## Android应用结构分析

---

- `app`
  - `build`:存放该项目的构建结果
  - `libs`:存放该项目依赖的第三方类库
  - `src` : 存放源代码和资源
    - `androidTest` : 代表了Android测试项目
    - `main`
      - `java` ( 存放Java或Kotlin源文件)
      - `res` ( 存放资源 )
        - `drawable`
        - `layout`
        - `mipmap-xxx`
        - `values`
      - `AndroidManifest.xml`
    - `test`
  - `build.gradle` (Gradle构建文件)
  - `.gitignore` ( 该文件配置Git工具忽略管理哪些文件)

## 简述

- `lib` : Android Studio默认会从国外的中央仓库下载JAR包，如果不能顺畅的连接国外网络，将会报错
- `build\generated\source\r\`
  - Android Studio自动生成的R.java文件，R.java文件由AAPT工具根据应用中的资源文件自动生成
  - Android应用的资源字典
  - 生成规则
    - 每类资源都对应于R类的一个内部类。比如所有页面布局资源对应于`layout`内部类
    - 每个具体的资源项都对应于内部类的一个`public static final int`类型的字段。
    - 类似于散列表

## res

- 对应标准的Gradle程序叫做`resources`目录
- 存放Android项目的各种资源文件
- `layout` : 存放页面布局文件
- `values` : 存放各种XML格式的资源文件
  - `string.xml` : 字符串资源文件



- colors.xml:颜色资源文件
- dimens.xml:尺寸资源文件
- drawable:存放XML文件定义的Drawable资源
  - drawable-ldpi/mdpi/hdpi/xdpi/xxdpi

## 流程

1. //举例在res/value/string.xml定义一个字符串

```
<resources>
 <string name = "hello">helloWorld</string>
</resources>
```

2. //在java代码中使用  
R.string.hello

3. //在xml中使用  
@string/hello

- 注意，如果在XML文件中使用标识符则无须使用专门的资源定义，直接在XML文档中使用 @+id/<标识符代号> 形式即可

- //定义  
android:id="@+id/ok"  
//在xml中使用  
@id/ok

- //在Activity中使用  
findViewById(R.id.ok)

## AndroidManifest.xml

- 控制Android应用的名称、图标、访问权限和包名等整体属性，包名将作为应用的唯一标识，
- **Android应用的Activity、Service、ContentProvider和BroadcastReceiver组件都需要在该文件中配置**
- 应用程序兼容的最低版本
- 应用程序使用系统所需的权限声明
  - 一个Android应用可能需要权限才能调用Android系统的功能，因此他也需要声明调用自身所需要的权限
- ```
<uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
```
- 其他程序访问该程序所需的权限声明

基本组件和通信

- Android应用通常由一个或多个基本组件组成

Activity

- **单个应用的单个组件的业务逻辑**
- Activity是Android应用中负责与用户交互的组件，使用setContentView(View)来显示指定组件
- Activity是Window的容器，Activity包含一个getWindow()方法，该方法返回该Activity所包含的窗口
 - **问题**：有setWindow方法，设置Activity的可视化用户界面吗？如果没有，为什么要那样设计？（无网，有网之后查询）
- 如果需要多个用户界面，那么Android应用会包含多个activity，多个Activity组成Activity栈，当前活动的Activity位于栈顶

View

- View组件是所有UI控件、容器控件的基类，View组件就是Android应用中用户实实在在看到的部分。
- View组件需要放到容器组件，或者使用Activity将它显示出来

Service

- Service与Activity的地位是并列的，它也代表一个单独的Android组件。
- Service与Activity的区别：Service通常位于后台运行，它一般不需要与用户交互，因此Service组件没有图形用户界面

BroadcastReceiver

- 广播信息接收器
- **单个应用的多个组件之间的数据交互**
- 类似于时间编程中的事件监听器
- BroadcastReceiver与**普通**事件监听器的区别
 - 普通事件监听器监听的事件源是程序中的对象
 - BroadcastReceiver监听的事件源是Android应用中的其他组件，相当于一个全局的事件监听器
- 使用
 1. 实现BroadcastReceiver子类，重写onReceive (Context context, Intent intent) 方法
 2. 其他组件通过sendBroadcast ()、sendStickyBroadcast () 或sendOrderedBroadcast () 方法发送广播信息
 3. 如果该BroadcastReceiver能对广播信息响应（通过IntentFilter配置），该BroadcastReceiver的onReceive (Context context, Intent intent) 方法将会被触发
- 注册
 - 在java代码中通过Context.registerReceiver()方法注册BroadcastReceiver
 - 在AndroidManifest.xml文件中使用<receiver.../>元素完成注册

ContentProvider

- **多个应用间进行数据交互**
- 当实现ContentProvider时，需要实现的抽象方法
 - 类似于SQL的增删改查
 - insert (Uri, ContentValues)
 - delete (Uri, ContentValues)
 - update (Uri, ContentValues, String, String[])

- query (Uri , String[] , String , String[] , String)
- 与之结合使用的是ContentResolver,应用程序使用ContentProvider暴露自己的数据，通过ContentResolver来访问数据
 - **问题**：多进程，共享数据，是如何保证事务的ACID和怎么进行同步的？（无网，有网之后查询）

Intent和IntentFilter

- Android应用内不同组件之间通行的载体
- Intent可以启动应用中另一个Activity，也可以启动一个Service组件，还可以发送一条广播消息来触发系统中的BroadcastReceiver
- **Activity、Service、BroadcastReceiver三种组件之间的通信都以Intent作为载体，只是不同组件使用Intent的机制略有不同**
- 分类
 - 启动Activity，可调用Context的startActivity (Intent intent) 或startActivityForResult (Intent intent , int requestCode)，这两个方法中的Intent参数封装了需要启动的目标Activity的信息
 - 启动Service，可调用Context的startService (Intent intent) 或bindService (Intent service , ServiceConnection conn , int flags)，这两个方法中的Intent参数封装了需要启动的目标Service的信息
 - 启动BroadcastReceiver，可调用Context的sendBroadcast (Intent intent) 或sendStickyBroadcast (Intent service) 或sendOrderedBroadcast (Intent , String receiverPermission) 这两个方法中的Intent参数封装了需要启动的目标Service的信息
- 当一个组件通过Intent表示了启动或触发另一个组件的“意图”，该意图可分为两类
 - 显式Intent：明确指定需要启动或触发的组件的类名
 - 隐式Intent：指定需要启动或触发的组件应满足怎样的条件
 - 判断是否符合隐式Intent
 - 使用IntentFilter来实现
 - 被调用组件通过IntentFilter来声明自己所满足的条件

签名APK

- Android项目以它的包名作为唯一标识，如果安装两个包名一致的应用，后安装的将会覆盖前面安装的
- 作用
 - 确定发布者的身份。
 - 确保应用的完整性。

总结

1. Android是什么
2. Android是干什么的
3. Android默认使用gradle作为项目管理工具，需掌握Gradle
4. 学习IDE的使用，Android Studio，Android SDK，以及Android SDK提供的模拟器和调试器
5. 如何创建Android应用
6. Android应用的目录结构，初步了解其作用
7. 初步了解基本组件Activity，Service，BroadcastReceiver，ContentProvider以及单个应用的组件间通信的载体Intent和IntentFilter。

| XML属性 | 说明 | 值（都支持） | 说明 |
|-----------------------|-------------|-------------------------------|--|
| android:layout height | 指定该子组件的布局高度 | match_parent (早期叫fill parent) | 指定子组件的高度、宽度与父容器组件的高度、宽度相同(实际上还要减去填充的空白距离)。 |
| android:layout width | 指定该子组件的布局宽度 | wrap content | 指定子组件的大小恰好能包裹它的内容即可。 |

- 注意：Android 组件的大小不仅受它实际的宽度、高度控制，还受它的布局高度与布局宽度控制。比如设置一个组件的宽度为30pt, 如果将它的布局宽度设为match parent, 那么该组件的宽度将会被“拉宽”到占满它所在的父容器;如果将它的布局宽度设为wrap content, 那么该组件的宽度才会是30pt。

- ViewGroup.MarginLayoutParams用于控制子组件周围的页边距(Margin,也就是组件四周的留白)

| XML属性 | 相关方法 | 说明 |
|---------------------------------|------------------------------|-----------------|
| android:layout margin | setMargins(int,int,int,int) | 指定该子组件四周的页边距 |
| android:layout marginBottom | setMargins(int, int,int,int) | 指定该子组件下边的页边距 |
| android:layout marginEnd | setMargins(int,int,int,int) | 指定该子组件结尾处的页边距 |
| android:layout marginHorizontal | setMargins(int,int,int,int) | 指定该子组件左、右两边的页边距 |
| android:layout marginLeft | setMargins(int,int,int,int) | 指定该子组件左边的页边距 |

控制UI界面

- XML控制：使用XML布局文件将视图和逻辑分离开
 - 在activity显示
 - `setContentView(R.layout.<资源文件名字>)`
 - 当在布局文件中添加多个UI组件时，都可以为该UI组件指定android:id属性，该属性的属性值代表该组件的唯一标识。
 - `findViewById(R.id. <android.id属性值>)`
- java控制：虽然Android推荐使用XML布局文件来控制UI界面，但如果开发者愿意可以完全抛弃XML布局文件，完全在Java或Kotlin代码中控制UI界面。
 - 无论创建哪种UI组件，都需要传入一个this参数，这是由于创建UI组件时传入一个Context参数，Context 代表访问Android应用环境的全局信息的API。让UI组件持有一个Context参数，可以让这些UI组件通过该Context参数来获取Android应用环境的全局信息。

- Context本身是一个抽象类，Android应用的Activity、Service都继承了Context,因此Activity、Service都可直接作为Context使用。
- XML和java控制：当混合使用XML布局文件和代码来控制UI界面时，习惯上把变化小、行为比较固定的组件放在XML布局文件中管理，而那些变化较多、行为控制比较复杂的组件则交给代码来管理。

自定义View

- View组件只是一个矩形的空白区域，View组件没有任何内容。对于Android应用的其他UI组件来说，它们都继承了View组件，然后在View组件提供的空白区域绘制外观。
- 基于Android UI组件的实现原理，开发者完全可以开发出项目定制的组件——当Android系统提供的UI组件不足以满足项目需要时，开发者可以通过继承View来派生自定义组件。
- 方法：当开发者打算派生自己的UI组件时，首先要定义1个继承View基类的子类，然后重写View类的一个或多个方法。

| 方法 | 说明 |
|---------------------------------------|---|
| 构造器 | 重写构造器是定制View的最基本方式，当Java或Kotlin代码创建一个View实例，或根据XML布局文件加载并构建界面时将，需要调用该构造器。 |
| onFinishInflate() | 这是一个回调方法，当应用从XML布局文件加载该组件并利用它来构建界面之后，该方法将会被回调。 |
| onMeasure(int,int) | 调用该方法来检测View组件及其所包含的所有子组件的大小。 |
| onLayout(boolean, int, int, int, int) | 当该组件需要分配其子组件的位置、大小时，该方法就会被回调。 |
| onDraw(Canvas) | 当该组件将要绘制它的内容时回调该方法。 |
| onSizeChanged(int, int, int, int) | 当该组件的大小被改变时回调该方法。 |
| onKeyDown(int, KeyEvent) | 当某个键被按下时触发该方法。 |

- 注意：
 - 当需要开发自定义View时，开发者并不需要重写上面列出的所有方法，而是可以根据业务需要重写上面的部分方法。
 - 如果要新建的自定义组件只是组合现有组件，不需要重新绘制所有组件内容，只要实现自定义组件的构造器,在自定义构造器中使用LayoutInflater加载布局文件即可。

布局管理器

- 背景：不同手机屏幕的分辨率、尺寸开不完全相同，如果让程序手动控制每个组件的大小、位置，则将给编程带来巨大的困难。
- 目的：让组件在不同的手机屏幕上都能运行良好
- 作用：布局管理器可以根据运行平台来调整组件的大小
- Android的布局管理器本身就是一个UI组件，所有的布局管理器都是ViewGroup的子类。
 - 所有布局都可作为容器类使用，因此可以调用多个重载的addView()向布局管理器中添加组件。

- 可以将一个布局管理器嵌套到其他布局管理器中，因为布局管理器也继承了View,也可以作为普通UI组件使用。

线性布局

- 线性布局由LinearLayout类来代表，它们都会将容器里的组件**一个挨着一个排列**起来。Linearl ayout可以控制各组件横向排列(通过设置android:orientation属性控制)，也可控制各组件纵向排列。
- Android的线性布局**不会换行**，当组件一个挨着一个排列到头之后，**剩下的组件将不会被显示出来**。

| XML属性 | 相关方法 | 说明 |
|---------------------------------|--|---|
| android:baselineAligned | setBaselineAligned(boolean) | 该属性设为false, 将会阻止该布局管理器与它的子元素的基线对齐 |
| android:divider | setDividerDrawable(Drawable) | 设置垂直布局时两个按钮之间的分隔条 |
| android:gravity | setGravity(int) | 设置布局管理器内组件的对齐方式。该属性支持top、bottom、left、right、center_vertical、fill_vertical、center_horizontal、fill_horizontal、center、fill、clip_vertical、clip_horizontal 几个属性值。也可以同时指定多种对齐方式的组合，例如left center_vertical 代表出现在屏幕左边，而且垂直居中 |
| android:measureWithLargestChild | setMeasureWithLargestChildEnabled(boolean) | 当该属性设为true时，所有带权重的子元素都会具有最大子元素的最小尺寸 |
| android:orientation | setOrientation(int) | 设置布局管理器内组件的排列方式，可以设置为horizontal (水平排列)、vertical (垂直排列，默认值)两个值的其中之一 |
| android:weightSum | | 设置该布局管理器的最大权值和 |

- LinearLayout包含的所有子元素都受LinearLayout.LayoutParams 控制，因此LinearLayout包含的**子元素**可以额外指定属性。

| ○ | XML属性 | 说明 |
|---|------------------------|---------------------------|
| | android:layout_gravity | 指定该子元素在LinearLayout中的对齐方式 |
| | android:layout_weight | 指定该子元素在LinearLayout中所占的权重 |

表格布局

- 表格布局由TableLayout所代表，**TableLayout 继承了LinearLayout**,因此它的本质依然是线性布局管理器。
- 表格布局**采用行、列的形式**来管理UI组件，TableLayout并不需要明确地声明包含多少行、多少列，而是通过添加TableRow、其他组件来控制表格的行数和列数。
- 每次向TableLayout中添加一个TableRow,该TableRow就是一个表格行，TableRow也是容器，因此它也可以不断地添加其他组件，每添加一个子组件该表格就增加一列。
- 如果直接向TableLayout中染加组件，那么这个组件将直接占用一行。
- 在表格布局中，列的宽度由该列中最宽的那个单元格决定，整个表格布局的宽度则取决于父容器的宽度(默认总是占满父容器本身)。
- 单元格行为
 - Shrinkable: 如果某个列被设为Shrinkable, 那么该列的所有单元格的宽度可以被收缩，以保证该表格能适应父容器的宽度。
 - Stretchable: 如果某个列被设为Stretchable,那么该列的所有单元格的宽度可以被拉伸，以保证组件能完全填满表格空余空间。
 - Collapsed: 如果某个列被设为Collapsed,那么该列的所有单元格会被隐藏。
- TableLayout继承了LinearLayout,因此它完全可以支持LinearLayout所支持的全部XML属性。除此之外，TableLayout还有自己独特的方法

| ○ | XML属性 | 相关方法 | 说明 |
|---|-------------------------|---------------------------------|---------------------------------|
| | android:collapseColumns | setColumnCollapsed(int,boolean) | 设置需要被隐藏的列的列序号。多个列序号之间用逗号隔开，从0开始 |
| | android:shrinkColumns | setShrinkAllColumns(boolean) | 设置允许被收缩的列的列序号。多个列序号之间用逗号隔开，从0开始 |
| | android:stretchColumns | setStretchAllColumns(boolean) | 设置允许被拉伸的列的列序号。多个列序号之间用逗号隔开，从0开始 |

帧布局

- 帧布局由FrameLayout所代表，**FrameLayout 直接继承了ViewGroup** 组件。
- 帧布局容器为每个加入其中的组件都创建一个空白的区域(称为一帧), 每个子组件占据1帧，这些帧都会根据gravity 属性执行自动对齐。帧布局的效果有点类似于AWT编程的CardLayout,都是**把组件——一个个叠加在一起**。与CardLayout的区别在于，CardLayout可以将下面的Card移上来，但FrameLayout则没有提供相应的方法。

| XML属性 | 相关方法 | 说明 |
|----------------------------|--------------------------------|---|
| android:foregroundGravity | setForegroundGravity(int) | 定义绘制前景图像的gravity 属性 |
| android:measureAllChildren | setMeasureAllChildren(boolean) | 设置该布局管理器计算大小时是否考虑所有子组件，默认为false,即只考虑可见状态的组件 |

- FrameLayout包含的子元素也受FrameLayout.LayoutParams控制，因此它所包含的子元素也可指定android:layout_gravity 属性，该属性控制该子元素在FrameLayout中的对齐方式。
- Android的View和UI组件不是线程安全的，所以Android不允许开发者启动线程访问用户界面的UI组件。

绝对布局

- 绝对布局由AbsoluteLayout所代表。**AbsoluteLayout继承自LinearLayout**
- Android不提供任何布局控制，而是由开发人员自己通过X、Y坐标来控制组件的位置。当使用AbsoluteLayout作为布局容器时，布局容器不再管理子组件的位置、大小— 这些都需要开发人员自己控制。
- 注意：使用绝对布局会很难兼顾不同屏幕大小、分辨率的问题。因此，AbsoluteLayout 布局管理器已经过时。

| XML属性 | 说明 |
|------------------|-------------|
| android:layout_x | 指定该子组件的X坐标。 |
| android:layout_y | 指定该子组件的Y坐标。 |

- Android常用的距离单位。
 - px (像素) :每个px对应屏幕上的一个点。
 - dip或dp (device independent pixels,设备独立像素) :-种基于屏幕密度的抽象单位。在每英寸160点的显示器上，ldip= lpx。但随着屏幕密度的改变，dip与px的换算会发生改变
 - sp (scaled pixels,比例像素) :主要处理字体的大小，可以根据用户的字体大小首选项进行缩放。
 - in (英寸) :标准长度单位。
 - mm(毫米):标准长度单位。
 - pt (磅) :标准长度单位， 1/72英寸。

约束布局

- 从功能上讲，约束布局相当于相对布局的改进
- 背景：相对布局只能控制组件在父容器中居中，或者与父容器(或另一个组件)左对齐、右对齐、顶端对齐、底端对齐，或者控制组件在另一个组件的左边、右边、上方或下方。但如果要控制组件位于父容器的某个百分比处，或者控制组件位于另一个组件的左边25dp处，相对布局就无能为力了，此时就可借助于约束布局。
- 使用Android Studio 新建项目时添加的Activity 的默认布局文件自动使用约束布局
- 约束布局中的组件的layout_width、layout_height属性值
 1. wrap content: 该组件的宽度或高度刚好显示组件内容。
 2. 固定长度值:设置该组件的宽度或高度为固定值。
 3. match constraint (0dp) :该组件的宽度或高度完全按约束计算。
- 设置垂直方向的位置的百分比为5%

- `app: layout_ constraintVertical_bias="0.05"`

| XML属性 | 说明 |
|----------------------------------|--|
| app:layout_constraintXxx_toXxxOf | 设置该组件与父容器(或其他组件)对齐。其中Xxx可以是Start、End、Top、Bottom 等值，具体可根据实际情况设置。该属性的属性值可以是parent (代表父容器)或其他组件的ID。 |
| android:layout_marginXxx | 设置该组件与参照组件的间距。其中Xxx可以是Start、End、Top、Bottom等值，具体可根据实际情况设置。 |
| app:layout_constraintXxx_bias | 设置该组件在参照组件中的百分比。其中Xxx可以是Horizontal或Vertical。 |

- 约束布局还提供了两种自动创建约束的方式。
 - 自动连接(Autoconnect)：。自动连接默认是关闭的，可以通过单击约束布局设计器上方的Autoconnect图标来打开自动连接。
 - 如果打开了约束布局的自动连接，那么每次向约束布局中拖入新组件时，约束布局设计器都会自动为该组件添加上、下、左、右4个约束——并不保证所添加的约束符合开发者的期望，因此通过自动连接添加的约束通常都需要手动修改。
 - 推断(Infer) :可以通过单击约束布局设计器上方的Infer图标执行推断。
 - “推断”功能则是另一种用法:用户先向界面中添加所有的UI组件，再通过拖曳摆放这些组件的位置，然后单击约束布局设计器上方的Infer 图标执行推断，约束布局设计器会自动为所有组件添加上、下、左、右4个约束——依然不保证所添加的约束符合开发者的期望。

引导线

- 背景：有时候，一批组件都需要放在容器的某个位置，或者某个组件需要参照的组件并不存在，为此约束布局为之提供了引导线(Guideline)。
- 介绍：引导线是约束布局中不可见的、却实实在在存在的组件，引导线唯一的作用就是被其他组件作为定位的参照。
- 种类:水平引导线和垂直引导线。
- 引导线对应于Guideline元素，属性

- | XML属性 | 说明 |
|------------------------------------|--|
| android:orientation | 该属性指定引导线的方向。该属性支持vertical和horizontal两个值，代表垂直引导线和水平引导线。 |
| app:layout_constraintGuide_begin | 该属性指定引导线到父容器起始处的距离。该属性值可以是一个距离值。 |
| app:layout_constraintGuide_end | 该属性指定引导线到父容器结束处的距离。该属性值可以是一个距离值。 |
| app:layout_constraintGuide_percent | 该属性指定引导线位于父容器的指定百分比处。 |
- 注意：后3个属性的作用是相同的，只是代表引导线的3种不同的定位方式，它们都用于控制引导线的位置，因此只要指定其中之一即可。

分界线

- 约束布局还提供了分界线(Barrier)来定位其他组件，分界线同样是不可见的、但实实在在存在的组件。
- 分界线同样可分为水平分界线和垂直分界线两种

引导线和分界线的区别

- 引导线以父容器为基础进行定位，比如设置与父容器的起始处、结束处的距离来控制引导线的定位，也可通过设置位于父容器的百分比来控制引导线的定位；
- 分界线根据容器中的组件进行定位，分界线可位于多个组件的上方、下方、左边或右边。

TextView及其子类

文本框(TextView) 和编辑框(EditText)

- TextView直接继承了View,它还是EditText、Button 两个UI组件类的父类。
- TextView 的作用就是在界面上显示文本
- TextView其实就是一个文本编辑器，只是Android关闭了它的文字编辑功能。
 - 如果开发者想要定义一个可编辑内容的文本框，则可以使用它的子类: EditText, EditText 允许用户编辑文本框中的内容。
 - TextView还派生了一个CheckedTextView, CheckedTextView 增加了一个checked状态，开发者可通过setChecked(boolean)和isChecked()方法来改变、访问该组件的checked状态。该组件还可通过setCheckMarkDrawable()方法来设置它的勾选图标。
 - TextView 还派生出了Button类。

EditText

- EditText与TextView 非常相似，它甚至与TextView共用了绝大部分XML属性和方法。
- EditText与TextView的最大区别在于: EditText 可以接受用户输入。
- EditText组件最重要的属性是inputType,该属性相当于HTML的<input..>元素的type属性，用于将EditText设置为指定类型的输入组件。
- 子类

- `AutoCompleteTextView`:带有自动完成功能的`EditText`
- `ExtractEditText`: 它并不是UI组件，而是`EditText`组件的底层服务类，负责提供全屏输入法支持。

按钮(Button) 组件

- `Button`继承了`TextView`,它主要是在UI界面上生成一个按钮，该按钮可以供用户单击，当用户单击按钮时，按钮会触发一个`onClick`事件。