移动安全

Mobile Security

第二章　移动开发基础

王寅

天津大学　网络安全学院

wyin@tju.edu.cn

本章目录

Android 应用的组成

输入城市　天气结果　查询失败

- **代码**：处理点击、查询与界面切换。
- **资源**：提供布局、文字和天气图标。
- **依赖**：网络库处理连接和请求。

例：一个基础的 Android 工程

```
app/
  src/main/
    AndroidManifest.xml
    java/.../
      MainActivity.java
      WeatherClient.java
    res/layout/activity_main.xml
    res/values/strings.xml
    res/drawable/ic_sun.xml
```

- **MainActivity.java**：接收点击、更新界面。
- **WeatherClient.java**：请求并解析天气。
- **activity_main.xml**：排列控件。
- **strings.xml / ic_sun.xml**：提供文字和图标。
- **Manifest**：声明入口与权限。

F1 Java 文件：定义逻辑与事件处理

```
public class MainActivity
    extends Activity {
 @Override
 protected void onCreate(Bundle state) {
  super.onCreate(state);
  setContentView(R.layout.activity_main);
  Button query =
      findViewById(R.id.query);
  query.setOnClickListener(
      v -> onQuery());
 }
}
```

- **类与方法**：MainActivity 组织页面行为，onCreate 处理页面创建。
- **界面连接**：加载布局，再用 id 找到按钮。
- **注册回调**：登记点击监听，点击后执行 onQuery。

Kotlin：安卓官方开发语言，兼容 Java 语言，更加高效、易用。

例：读取城市并发起查询

```
void onQuery() {
  String city = cityInput.getText()
      .toString().trim();
  if (city.isEmpty()) return;
  status.setText(R.string.loading);
  client.fetch(city,
    data -> runOnUiThread(
        () -> showWeather(data)),
    error -> runOnUiThread(
        this::showError));
}
```

1. **读取输入**：取城市文字，去掉首尾空格，空值直接返回。
2. **发起查询**：显示“正在查询”，再异步请求。
3. **更新界面**：回到主线程，显示天气或失败提示。

F1.1 Java 特性：抽象

```
abstract class WeatherSource {
  abstract String query(String city);
}
class CachedWeather
    extends WeatherSource {
  @Override String query(String city) {
    return "26 C";
  }
}
```

- **概念**：抽象类不能直接实例化；抽象方法声明操作而不提供实现。
- **代码片段的解释**：WeatherSource 约定 query；CachedWeather 提供方法体。
- **对安全分析的影响**：抽象方法没有完整行为，分析要继续找到具体实现。

F1.2 Java 特性：封装

```
class WeatherRequest {
  private String city;
  public void setCity(String value) {
    if (value == null ||
        value.trim().isEmpty())
      throw new IllegalArgumentException(
          "empty city");
    city = value;
  }
  public String city() { return city; }
}
```

- **概念**：隐藏对象的内部状态，通过对象提供的方法与其交互。
- **代码片段的解释**：city 为 private；setCity 集中检查输入。
- **对安全分析的影响**：沿公开方法追踪数据。private 限制代码访问，无法替代加密或业务授权。

F1.3 Java 特性：继承

```
class BaseClient {
  String endpoint() {
    return "https://weather.example";
  }
}
class WeatherClient extends BaseClient {
  String target() {
    return endpoint();
  }
}
```

- **概念**：子类从父类继承字段和方法，并可扩展或重写行为。
- **代码片段的解释**：WeatherClient 未定义 endpoint，调用来自 BaseClient。
- **对安全分析的影响**：地址、校验或回调处理可能位于父类，要沿继承链查找。

F1.4 Java 特性：多态

```
class RemoteWeather
    extends WeatherSource {
  @Override String query(String city) {
    return "remote:" + city;
  }
}

WeatherSource source = useCache
    ? new CachedWeather()
    : new RemoteWeather();
String text = source.query("Tianjin");
```

- **概念**：子类可提供不同实现；调用被重写的实例方法时，由实际对象决定执行版本。
- **代码片段的解释**：useCache 决定对象类型，同一个 query 调用可得到不同结果。
- **对安全分析的影响**：追踪 new、工厂方法或依赖注入，确定运行时的调用目标。

F1.5 Java 特性：反射

```
Class<?> type = Class.forName(
    "demo.WeatherPlugin");
Object plugin = type
    .getDeclaredConstructor()
    .newInstance();
Method query = type.getMethod(
    "query", String.class);
Object text = query.invoke(
    plugin, "Tianjin");
```

- **概念**：运行时获取字段、方法和构造器信息，并在访问规则内操作它们。
- **代码片段的解释**：类名和方法名都是字符串，invoke 才执行 query。
- **对安全分析的影响**：直接调用关系可能缺失，要跟踪名称来源、类加载器和实际参数。

F2 布局 XML：定义控件及其属性

```
<Button
  android:id="@+id/query"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="@string/query" />
```

- **控件类型**：Button 表示可点击按钮。
- **id 与尺寸**：id 供代码查找，宽高决定控件占用空间。
- **资源引用**：@string/query 提供按钮文字。

例：输入、按钮与文本的绑定

```
<LinearLayout ...>
  <EditText
      android:id="@+id/city" ... />
  <Button
      android:id="@+id/query"
      android:text="@string/query" ... />
  <TextView
      android:id="@+id/status" ... />
</LinearLayout>
```

1. **输入城市**：用户在 city 中填写，代码从这里取值。
2. **点击查询**：query 按钮的监听器执行 onQuery。
3. **查看结果**：status 显示加载、成功或失败状态。

F3 资源 XML：集中管理界面文字

```
<resources>
  <string name="app_name">
    天气查询
  </string>
  <string name="query">
    查询天气
  </string>
</resources>
```

- **存放位置**：strings.xml 保存命名字符串。
- **引用方式**：XML 用 @string/query，Java 用 R.string.query。
- **语言适配**：相同资源名可提供不同语言的文字。

F4 Manifest：应用的入口

```
<uses-permission android:name=
  "android.permission.INTERNET" />
<activity android:name=".Detail"
  android:exported="true"
  android:permission="demo.weather.READ">
  <intent-filter>
    <action android:name="demo.VIEW" />
    ...
  </intent-filter>
</activity>
```

- **uses-permission**：申请 App 需要使用的权限。
- **activity**：声明界面入口，name 指向实现类。
- **intent-filter**：声明可接收的请求类型。
- **android:exported**：控制组件是否对其他 App 开放。
- **android:permission**：指定调用入口所需的权限。

B1 开发

工具	在天气 App 开发中的作用
Android Studio	编辑 Java/XML，组织构建，运行与调试应用
Android SDK	提供平台 API、构建工具及 adb 等设备工具
Android Virtual Device（AVD）	保存虚拟设备配置：设备型号、系统镜像等
Android Emulator	按照 AVD 配置启动一台可运行 App 的虚拟设备

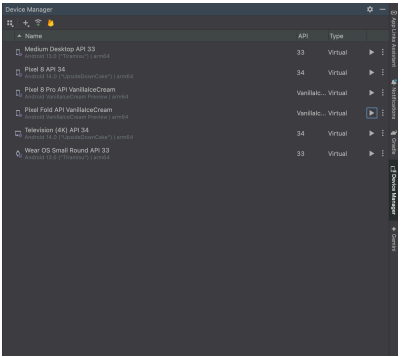
Android Studio

SDK Manager

Device Manager

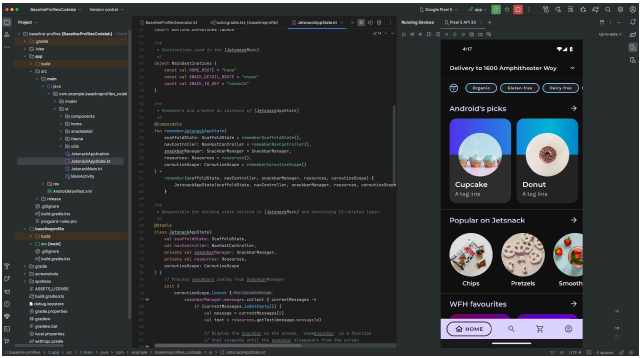
Emulator

测试：模拟器测试



1. **准备**：在 Device Manager 中选设备与系统镜像。
2. **运行**：启动模拟器，选择它作为 Run 目标。
3. **天气 App**：测试正常城市、空输入、断网重试和旋转。

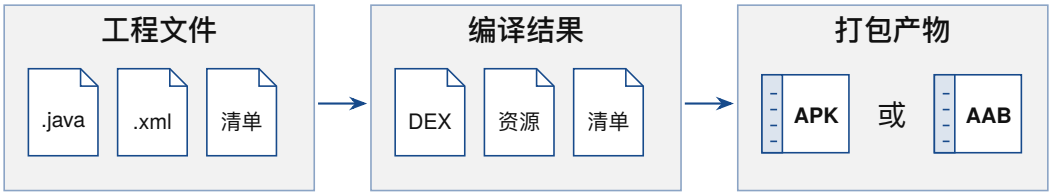
测试：真机测试



1. **连接**：开启开发者选项与 USB 调试，连接电脑并确认授权。
2. **运行**：选中真机，安装并启动 debug 版本。
3. **天气 App**：比较 Wi-Fi/移动网络、切到后台再返回。

B3 编译构建

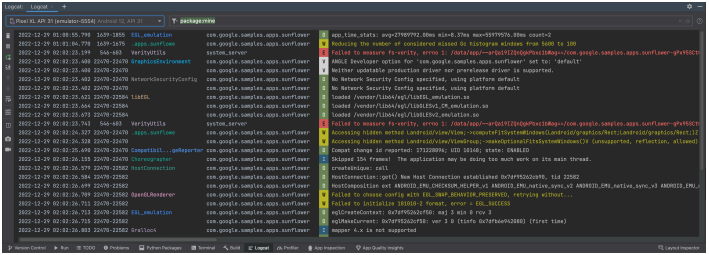
输入与处理	形成的内容
Java 源码及依赖 → 编译、DEX 转换	classes.dex 等可执行代码
资源 → 编译；Manifest → 合并	编译后的资源、资源表与最终清单
以上内容 → 打包、签名	用于设备安装的 APK，或用于发布的 AAB



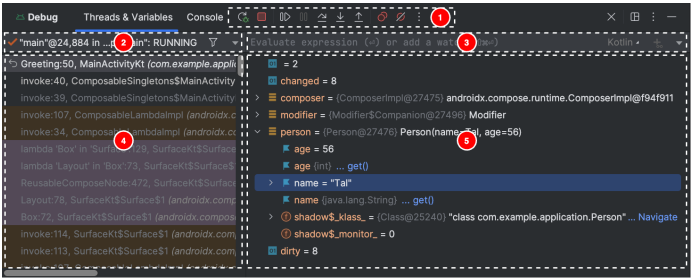
B4.1 调试：Logcat

Logcat 是一个命令行工具，可以捕获系统消息，包括应用的输出消息，以及设备上运行的所有应用和系统进程的日志消息。

开发者可以通过不同过滤层级的输出方式，查看不同级别的开发者日志。`Log.v()` 写入 VERBOSE 级别日志，`Log.e()` 写入 ERROR 级别日志。

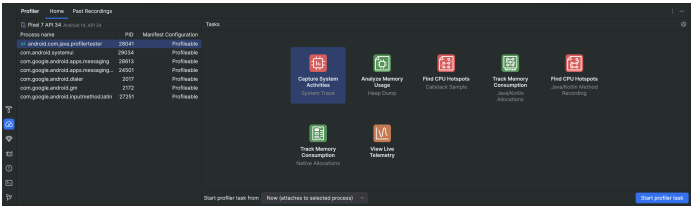


B4.2 调试：断点



1. **设置断点**：点击代码行号旁的边栏。
2. **Debug 运行**：命中断点后检查 Variables 与调用栈。
3. **逐步执行**：Step Over 看下一行，Step Into 进入方法。

B4.3 调试：性能分析



1. **选择任务**：在 Profiler 中选进程和分析任务。
2. **录制过程**：开始记录，执行待分析操作后停止。
3. **核对证据**：查看主线程等待和对象保留。

B5.1 发布：APK

```
weather.apk
  AndroidManifest.xml
  classes.dex
  resources.arsc
  res/
  assets/           (optional)
  lib/arm64-v8a/    (optional)
```

- **代码**：classes.dex 包含编译后的 Java/Kotlin 及依赖代码。
- **资源和清单**：res 与资源表保存界面材料；清单通常为编译形式。
- **可选内容**：assets 保存原始文件；lib 按设备架构放置.so。

APK 采用 ZIP 容器；解包可看文件，但代码与清单通常需要专门工具解析。

B5.2 发布：AAB

推行背景：2018 年 5 月推出，2021 年 8 月起 Google Play 要求新应用采用；旨在按设备分发所需内容，减小下载体积并简化多 APK 发布。

发布路线：AAB → 分发平台 / bundletool → 设备适配的 APK 集合 → 安装

注意：AAB 本身不能直接安装到设备；Google Play 用它生成实际交付的 APK。

APK	Android App Bundle（AAB）
交给设备安装	交给分发平台，或交给 bundletool 处理
含本次安装需要的代码与资源	按模块组织编译代码、资源及元数据
可作为独立包或一组拆分包分发	可生成适配设备架构、语言等条件的 APK

B999 生成

帮我开发一个 App。

工程构建小结

安全分析的目标	对应证据位置
点击按钮后会执行什么	布局 id、Java 回调与具体方法
有什么能力，哪些组件能被调用	最终 Manifest 与具体实现
某次操作到底执行了什么	测试条件、Logcat 日志与断点现场
最终交付了哪些内容	APK/AAB、依赖、构建配置与拆分包

R1 Application：进程初始化

Application 保存应用进程级的公共对象，例如网络客户端或依赖容器。

位置	适合放什么
Manifest 的 application	可通过 android:name 指定自定义 Application 类。
Application.onCreate()	快速完成必要初始化；较重的工作考虑按需加载。
Activity / Fragment	创建界面，并使用已经准备好的公共依赖。

检查初始化入口，有助于发现用户打开页面前已经执行的 SDK 代码。

R2 Android 四大组件

组件	承担的角色	直观例子
Activity（活动）	承载用户交互入口	查看一张照片
Service（服务）	提供无界面的服务功能	音乐播放服务
BroadcastReceiver（广播接收器）	响应特定广播事件	响应充电连接事件
ContentProvider（内容提供者）	通过约定接口提供数据	按 URI 读取获准分享的文件

默认情况下，四大组件可在同一进程运行；组件类型不决定进程或线程。

R2.1 Activity：用户交互入口

职责：承载窗口，响应用户操作，由系统管理前后台状态。

例如，一个社交应用可能包含好友列表的 Activity，展示消息的 Activity，以及编辑个人资料的 Activity 等。

注意：安卓应用的每一个 Activity 都要在 AndroidManifest.xml 清单文件中声明。

在安全分析中的用途：是程序的交互入口点，以此进入内部开展行为分析。

R2.1 Activity 生命周期

(1) **活动创建**：初始化，包括布局设置、初始化数据等，此时用户不可见；

(2) **活动开始**：活动可见，但是无法互动；

(3) **活动运行**：进入 Resumed 状态，可以参与前台交互；

问题

系统会调用哪个生命周期方法来显示 activity?

- onPause()
- onVisible()
- onStart()
- onDestroy()

问题

系统会调用哪个生命周期方法来显示 activity?

- onPause()
- onVisible()
- onStart()
- onDestroy()

解析

答案：onStart()。进入 Started 状态后，Activity 对用户可见。onResume 进一步对应前台交互状态；跟踪回调有助于复现某段代码的执行时机。

例：打开详情再返回

设定：普通单窗口；列表 Activity A 存活；新建全屏 Activity B；返回时结束 B。

用户动作	正常路径中的回调顺序
A 打开 B	A.onPause → B.onCreate → B.onStart → B.onResume → A.onStop
结束 B，返回 A	B.onPause → A.onRestart → A.onStart → A.onResume → B.onStop → B.onDestroy

A 返回时会再次 onResume；在本例条件下，无需再次 onCreate。

R2.2 Service：职责与功能

职责：处理长时间执行且不需要用户反复交互的服务。

例如，播放音乐（前台服务），下载文件（后台服务）等。

```
Intent task = new Intent(
    this, SyncService.class);
startService(task);
bindService(task,
    connection,
    Context.BIND_AUTO_CREATE);
```

- **启动功能**：Activity 等组件可以启动服务；启动后的服务可继续运行，Activity 销毁不会自动停止它。
- **绑定功能**：组件通过连接访问服务接口，与服务交互，也可支持进程间通信（IPC）。

R2.2 Service：限制与安全分析

平台限制：为了**改善续航、内存占用与用户体验**，Android 8 起后台服务受限；Android 12 起后台启动前台服务也受限制。

在安全分析中的作用：如果一个导出的 Service 没有做严格的限制，其他应用可能请求启动服务，或访问其提供的绑定接口。取决于被暴露的功能，应用可以去执行未授权的行为，获取敏感信息或污染修改内部应用的状态造成威胁。

R2.3 BroadcastReceiver：事件与回调

```
public class BatteryReceiver
    extends BroadcastReceiver {
  @Override
  public void onReceive(
      Context c, Intent i) {
    if (Intent.ACTION_BATTERY_LOW
        .equals(i.getAction())) {
      Log.i("Battery", "Low");
    }
  }
}
```

职责：用于监听和响应广播消息。

例如，响应电池电量低、屏幕关闭等（系统消息），响应数据更新（应用消息）等。

主要使用场景：系统事件的监听、应用间的数据交换和通信、定时任务等；

在安全分析中的作用：发送广播时，如果处理不当，恶意应用便可以嗅探，拦截广播，致使敏感数据泄露，接收广播时处理不当，便会导致拒绝服务攻击、伪造消息、越权操作。

R2.4 ContentProvider：数据访问入口

```
public class NotesProvider
    extends ContentProvider {
  @Override
  public Cursor query(Uri uri,
      String[] cols, String filter,
      String[] args, String order) {
    MatrixCursor c = new MatrixCursor(
        new String[]{"title"});
    c.newRow().add("Note");
    return c;
  }
  // 其余回调略
}
```

职责：用于数据存取及数据共享，抽象了数据的存储方式而无需关心存储细节。

例如，通过 URI 统一资源标识符来访问数据，对数据进行查询、插入、更新、删除等操作。

主要使用场景：隐藏数据存储细节、访问系统数据、跨应用共享等。

在安全分析中的作用：ContentProvider 的安全本质是“**可控的数据共享**”，共享范围超出预期就会带来数据泄漏威胁。

例：ContentProvider 路径穿越泄露私有文件

场景：Provider 对外导出，缺少读取权限校验；原本只打算共享 files/shared 中的文件。

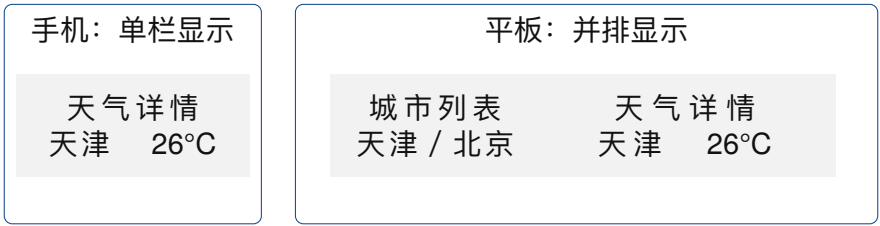
```
File root = getContext().getFilesDir();
File shared = new File(root, "shared");
String name =
    uri.getQueryParameter("name");
File target = new File(shared, name);
return ParcelFileDescriptor.open(
    target, MODE_READ_ONLY);
```

1. **输入**：调用方将 URI 中的 name 设为`../private.txt`。
2. **越界**：拼接结果指向`files/private.txt`，越过 shared 目录。
3. **后果**：Provider 以自身权限打开文件，向调用方返回可读句柄。

缓解措施：限制调用权限；按允许的文件标识映射路径，并核对规范化后的真实路径是否仍在共享目录内。

R3 Fragment：可复用的界面片段

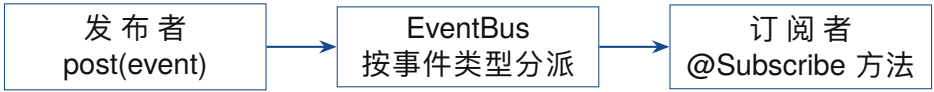
Fragment 是应用界面中可复用的一部分，由 Activity 或父 Fragment 承载。



区别：**Activity** 是独立的屏幕组件，而 **Fragment** 是依附于 **Activity** 的可复用 **UI** 模块。

R4 EventBus：进程内的事件发布与订阅

EventBus 是一个发布 / 订阅的事件总线。是发布者和订阅者模式，可以实现两个并不知晓的组件之间进行相互通信。可以在同一进程内向不同线程发布事件。



注册范围：按使用范围 register / unregister；普通事件发出时，订阅者须已注册。

通信选择：与 Android 系统相关的通知使用系统广播；本地广播（LocalBroadcastManager）仅在应用进程内分发，已弃用。

EventBus 不需要指定 Context，调度灵活，使用简单，开发轻量，适合与系统交互较少的同进程通信，例如 SDK 内部的事件通知。

应用组件小结

知识点	安全分析时要追问什么
R1 Application	进程启动时初始化了什么，哪些代码会提前执行?
R2 四大组件	入口由谁触发，回调何时执行，谁有权访问?
R3 Fragment	Activity 内部的实际页面逻辑在哪里?
R4 EventBus	事件由谁发布、谁订阅，最终触发了什么操作?

C1 Intent：组件之间的请求

Intent 是一种消息传递对象，可用于向其他应用组件请求操作。

常见的使用场景包括：

- **启动 Activity**：将 Intent 传给 `startActivity()`，打开界面并携带附加数据 (extra)。例如，从列表进入详情页。
- **启动服务**：通过 `startService()` 请求服务执行工作；也可通过 `bindService()` 建立连接。例如，向播放服务发送播放请求。
- **传送广播**：通过 `sendBroadcast()` 或 `sendOrderedBroadcast()` 发布事件，由符合条件的 Receiver 处理。例如，通知数据已更新。

C1.1 显式 Intent 与隐式 Intent

显式 Intent：通过指定完整的 `ComponentName` 来指定哪个应用的哪个组件将满足该 intent。

```
// Target component
Intent intent = new Intent(
    this, DetailActivity.class);
// Data for the target
intent.putExtra("city", "Tianjin");
startActivity(intent);
```

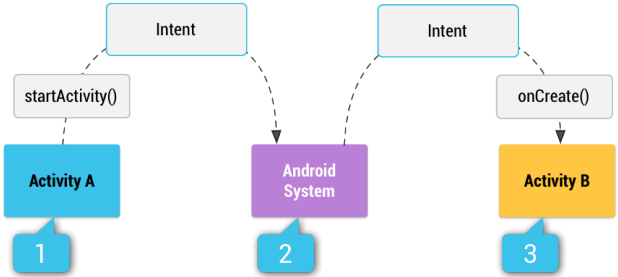
指定 DetailActivity，传入城市名。

隐式 Intent：不会指定特定组件，而是声明要执行的常规操作，从而允许其他应用的组件处理该 intent。

```
// Action and data
Uri page = Uri.parse(
    "https://example.org");
Intent intent = new Intent(
    Intent.ACTION_VIEW, page);
startActivity(intent);
```

由系统匹配可处理网页的 Activity。

C1.2 隐式 Intent 的系统分派



1. Activity A 创建 Intent 描述所需操作，调用 `startActivity()` 提交请求。
2. Android 系统查找与该 Intent 匹配的 `intent-filter`。
3. 找到目标 Activity B 后，调用其 `onCreate()`，并向它传递 Intent。

注意：为确保您的应用安全无虞，请务必在启动**Service**时使用显式 intent，并且不要为您的服务声明 intent 过滤器。使用隐式 intent 启动服务存在安全隐患，因为您无法确定哪些服务将响应 intent，且用户无法看到哪些服务已启动。从 Android 5.0（API 级别 21）开始，如果使用隐式 intent 调用**bindService()**，系统会抛出异常。

C1.3 构建 Intent

信息	作用	例子
组件名称	指定接收组件，形成显式请求。	`DetailActivity`：打开详情界面。
操作	说明希望执行什么动作。	`ACTION_VIEW`：查看内容。
数据	URI 表示处理对象；MIME 表示数据类型。	图片 URI 及 `image/png`。
类别	补充接收组件应具备的类别信息。	`CATEGORY_BROWSABLE`：允许由浏览器调用。
附加数据	以键值对传递业务参数。	`putExtra("city", " 天津")`。
标志	告诉系统如何启动或处理组件。	`FLAG_ACTIVITY_NEW_TASK`：影响 Activity 的任务归属。

隐式 Intent 的过滤器匹配主要检查操作、数据与类别；Extra 用于传参，不参与过滤器匹配。

例：隐式 Intent 攻击

发送方：令牌随隐式请求发出

```
Intent i = new Intent(
    "com.example.OPEN_DETAIL");
i.putExtra("token", sessionToken);
startActivity(i);
```

恶意接收方：读取会话令牌

```
Intent received = getIntent();
String token = received
    .getStringExtra("token");
```

代码片段的解释

1. 发送方用隐式请求传递会话令牌。
2. 恶意 Activity 注册同名动作与 DEFAULT 类别，参与匹配。
3. 若恶意组件被选中，即可读取令牌。

C2.1 广播的注册

静态注册（清单注册）

```
<receiver
    android:name=".UpdateReceiver"
    android:exported="false">
  <intent-filter>
    <action
        android:name=
          "com.example.UPDATE"/>
  </intent-filter>
</receiver>
```

系统根据清单登记接收器；满足条件时，可作为应用的独立入口。

动态注册（上下文注册）

```
// onStart() 中注册
receiver = new UpdateReceiver();
IntentFilter filter = new IntentFilter(
    "com.example.UPDATE");
ContextCompat.registerReceiver(
    this, receiver, filter,
    ContextCompat.RECEIVER_NOT_EXPORTED);
// onStop() 中解除注册
unregisterReceiver(receiver);
```

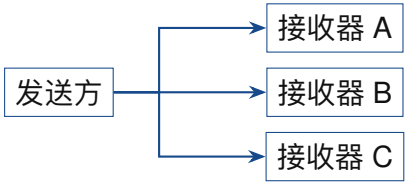
在有效注册期间接收广播；不再需要时，解除注册。

Android 8 起（目标 API 26 及以上），大部分隐式广播不再允许清单注册。

C2.2 广播的发送

标准广播（常规广播）

```
sendBroadcast(intent);
```



各接收器独立处理，顺序不保证；不传递接收结果，不能中止后续投递。

有序广播

```
sendOrderedBroadcast(
intent, permission);
```

接收器逐个处理，可传递结果；在允许中止时，可阻止后续接收器处理。

例：粘性广播缓存会话信息的风险

粘性广播会在发送完成后保留 Intent，供之后注册的接收器获取。它曾用于补发当前状态，相关发送 API 已在 Android 5.0（API 21）弃用。

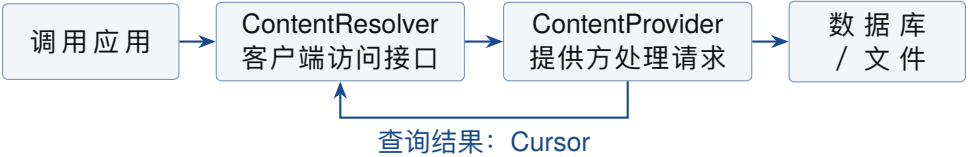
```
// 旧式发送端
Intent state = new Intent(
    "com.example.SESSION");
state.putExtra("token", "demo-token");
sendStickyBroadcast(state);

// 应用查询缓存消息
Intent cached = registerReceiver(
    null, new IntentFilter(
        "com.example.SESSION"));
```

1. **保留**：发送方将会话令牌写入 extra；发送完成后，消息仍可能留在缓存中。
2. **泄露**：后来匹配该广播的应用，也可能取得令牌。
3. **篡改**：恶意应用可用匹配的 Intent 覆盖缓存中的 extra，令接收方读到伪造状态。

缓解措施：使用非粘性广播通知变化，通过受控接口查询当前状态；同进程状态可用 Flow 或 LiveData 传递。

C3.1 ContentProvider 的基本工作方式



客户端 API	操作与结果
`query()`	查询；返回 Cursor
`insert()`	插入；返回新记录 URI
`update()`	更新；返回影响行数
`delete()`	删除；返回影响行数

1. 调用方提交 URI、字段和筛选条件。
2. 系统定位 Provider，由它检查请求并操作底层存储。
3. 调用方从 `Cursor` 逐行读取查询结果，用完后关闭。

对安全分析的影响：沿调用关系找到 Provider 处理方法，确认数据的读取者与修改者。

C3.2 内容 URI

content://user_dictionary/words/4

协议：content　提供方标识：user_dictionary　数据路径：/words/4

```
Uri tableUri = Uri.parse(
    "content://user_dictionary/words");
Uri rowUri = ContentUris.withAppendedId(
    tableUri, 4);
try (Cursor c = getContentResolver().query(
    rowUri, new String[]{"word"},
    null, null, null)) {
  if (c != null && c.moveToFirst()) {
    String word = c.getString(0);
  }
}
```

1. **找到用户字典**：user_dictionary 指定字典 Provider，words 指定单词集合。
2. **选中一条记录**：在地址末尾加上/4，指定 ID 为 4 的记录。ID 是记录的编号。
3. **取出单词**：query() 只查询 word 列；getString(0) 读出这一列的单词。

最终效果：有读取权限且记录存在时，将这条记录的单词保存到变量 word 中。

对安全分析的影响：改变记录 ID 后，检查调用者是否有权访问新的记录。

C3.3 Provider 的权限控制

按 Provider 控制读写

```
<provider
    android:name=".NotesProvider"
    android:authorities="demo.notes"
    android:exported="true"
    android:readPermission="demo.READ"
    android:writePermission="demo.WRITE"
/>
```

- `exported` 控制一般跨应用访问，导出后仍受权限限制。
- `readPermission` 与 `writePermission` 分别限制读取和修改。
- 调用方声明 `uses-permission`，按权限级别取得授权。

按选定 URI 临时授权

```
Intent result = new Intent();
result.setData(selectedUri);
result.addFlags(
    FLAG_GRANT_READ_URI_PERMISSION);
setResult(RESULT_OK, result);
finish();
```

- 返回选定附件的 URI，仅授予读取能力。
- 提供方须配置 `grantUriPermissions` 或 `grant-uri-permission`。
- 临时授权也可用于未导出的 Provider。

对安全分析的影响：检查谁能访问、能读写哪些数据，以及授权何时失效。

数据流转小结

- C1：Intent 的机制，显式 Intent 与隐式 Intent；
- C2：广播的注册与接收；
- C3：Provider 的工作方式与权限控制。

R1 NDK：使用 C / C++ 开发 Android 功能

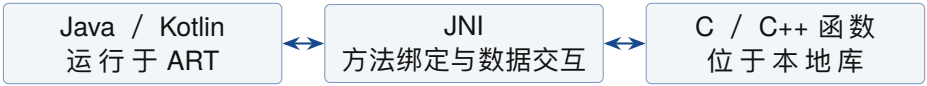
NDK（Native Development Kit）是支持 Android 本地开发的工具集，可将 C / C++ 代码编译为本地库。

- **产生原因**：复用已有 C / C++ 库，或满足音视频、图像处理等计算与低延迟需求。
- **职责划分**：Java / Kotlin 组织界面与业务；本地库完成选定功能；JNI 连接两侧调用。

例如：相机 App 在 Java 层响应按钮，通过 JNI 调用 C++ 图像处理库，再显示处理结果。

R1 JNI：跨语言调用的基本机制

JNI（Java Native Interface）规定受管理代码与本地代码如何调用、传参和返回结果。



1. **加载库**：`System.loadLibrary("sample")` 加载 `libsample.so`。
2. **绑定方法**：把 `native` 声明关联到 C / C++ 函数，可按名称查找或使用 `RegisterNatives` 注册。
3. **交换数据**：通过 JNI 类型和接口传递参数、访问对象，再将结果返回调用方。

对安全分析的影响：遇到 native 方法后继续进入.so；JNI 本身不建立权限隔离。

例：Java 声明本地加法方法

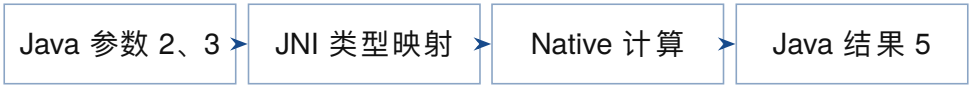
```java
package edu.course;
class NativeMath {
    static { System.loadLibrary("sample"); }
    static native int add(int a, int b);
}
// At another call site:
// int result = NativeMath.add(2, 3);
```

Java 侧声明方法与库名，本地实现负责具体计算。

例：JNI 传参与结果返回

```
extern "C" JNIEXPORT jint JNICALL
Java_edu_course_NativeMath_add(
    JNIEnv* env, jclass type, jint a, jint b) {
    return a + b;
}
```

JNI（Java 本地接口）连接 Java 与本地实现。

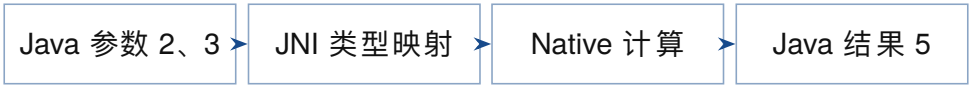


Java 调用时传入 2 和 3。

例：JNI 传参与结果返回

```
extern "C" JNIEXPORT jint JNICALL
Java_edu_course_NativeMath_add(
    JNIEnv* env, jclass type, jint a, jint b) {
    return a + b;
}
```

JNI（Java 本地接口）连接 Java 与本地实现。

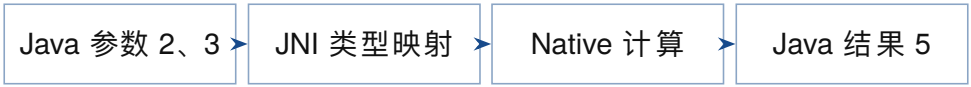


JNI 将参数连接到对应本地方法。

例：JNI 传参与结果返回

```
extern "C" JNIEXPORT jint JNICALL
Java_edu_course_NativeMath_add(
    JNIEnv* env, jclass type, jint a, jint b) {
    return a + b;
}
```

JNI（Java 本地接口）连接 Java 与本地实现。



本地计算得到 5，结果返回 Java 调用方。

例：JNI 本地库中的内存安全漏洞

Luca Di Bartolomeo 等 | USENIX Security 2025

Hercules Droidot and the murder on the JNI Express

- **研究什么**：有些 App 通过 JNI 调用 C/C++ 编写的本地库。论文检查这些库在处理数据时，是否会用错内存并导致崩溃。
- **怎么检查**：工具 POIROT 先从 App 代码中找出库的调用方式，再模仿这些调用，反复更换输入，寻找会引发崩溃的情况。
- **发现什么**：研究测试了 3,967 个含本地库的 App；从崩溃结果中抽查 200 个，确认 25 个缺陷，涉及 16 个本地库。同一个库被多个 App 使用时，缺陷可能影响多个 App。

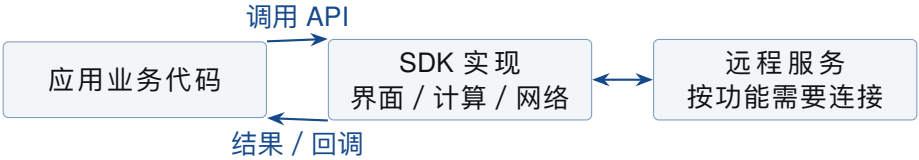
R2 SDK：复用第三方功能与服务

SDK（Software Development Kit，软件开发工具包）将接口、实现、文档与示例组合起来，帮助应用接入一项功能或服务。

- **产生原因**：广告、地图、支付等功能需要专门实现与持续维护；统一接口减少重复开发和接入差异。
- **提供内容**：Android 功能 SDK 常以 AAR 等库交付，可包含代码、资源、Manifest 声明与本地库。
- **使用方式**：应用配置 SDK、调用其 API，并通过返回值或回调获得处理结果。

例如：资讯 App 调用广告 SDK 展示广告，由 SDK 处理请求、素材展示与事件通知。

R2.1 SDK：从接入到执行



1. **接入与配置**：代码和资源参与构建；库的 Manifest 声明按规则合并到最终应用。
2. **初始化**：设置应用标识、运行参数与回调；部分 SDK 也会通过组件自动初始化。
3. **调用与反馈**：业务代码调用 API，SDK 执行功能或请求服务，通过回调报告结果。

对安全分析的影响：传统随 App 打包的 SDK 通常使用宿主进程及已获准的能力；需要一并检查其配置和行为。

例：资讯 App 接入 AdMob 广告 SDK

1. **应用准备广告位**：资讯列表底部放置 SDK 提供的 `AdView`。
2. **SDK 请求广告**：向广告服务提交请求，接收素材并显示在广告位中。
3. **反馈展示与交互**：通过监听器报告加载、展示或点击等事件。

例：请求并展示横幅广告

前提：已配置 AdMob 应用 ID，完成所需同意流程与 SDK 初始化；以下代码在主线程执行。

```
AdView adView = new AdView(this);
adView.setAdUnitId(
    "ca-app-pub-3940256099942544/"
    + "9214589741");
adView.setAdSize(bannerSize);
container.addView(adView);
AdRequest request =
    new AdRequest.Builder().build();
adView.loadAd(request);
```

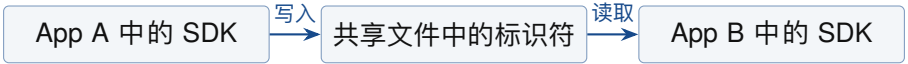
1. **创建与配置**：创建 `AdView`，设置官方测试广告单元 ID 和横幅尺寸。
2. **接入界面**：把广告视图添加到页面中的 `container`。
3. **发起请求**：构建 `AdRequest`，调用 `loadAd()`；结果由 SDK 异步处理。

例：第三方 SDK 的隐蔽标识符跟踪

Zikan Dong 等 | USENIX Security 2024

Exploring Covert Third-party Identifiers through External Storage in the Android New Era

- **研究什么**：有些 SDK 会把识别设备的一串编号存进共享文件。不同 App 里的 SDK 读到相同编号，就可能认出它们运行在同一台设备上。
- **怎么检查**：研究者自动操作 App，记录它们读写了哪些文件，再检查文件内容和代码，找出哪个 SDK 在保存、读取这些编号。
- **发现什么**：在 8,000 个 App 样本中识别出 17 个相关跟踪 SDK。论文还演示了把编号藏进媒体文件、由其他具备读取条件的 App 取出的做法。



R3.1 WebView：嵌入与加载网页

WebView 是一种显示网页的 Android 视图，可嵌入 Activity 布局。例如，在 App 内展示在线帮助、用户协议或活动页面。

布局 XML：为网页分配显示区域

```
<WebView
    android:id="@+id/help_web"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
```

Activity：取得视图并加载网址

```
WebView web =
    findViewById(R.id.help_web);
web.loadUrl(
    "https://www.example.com/help");
```

- **互联网连接**：Manifest 声明 `android.permission.INTERNET`。
- **界面职责**：WebView 负责呈现网页；地址栏、导航按钮由应用按需提供。

R3.2 WebView：在 WebView 中使用 JavaScript

启用 JavaScript

JavaScript 默认停用。通过 `getSettings()` 取得 `WebSettings`，再启用脚本执行。

```
WebView web =
    findViewById(R.id.help_web);
WebSettings settings = web.getSettings();
settings.setJavaScriptEnabled(true);
```

适用情形：页面使用 JavaScript 处理按钮事件、表单或动态更新内容。

将 JavaScript 代码绑定到 Android 代码

调用 `addJavascriptInterface()`，传入 Java 对象及网页使用的接口名。

```
web.addJavascriptInterface(
    new PageBridge(), "Android");
```

例：网页通过 `Android.showToast('Hello')` 调用对象的方法，显示 Android 提示消息。

供网页调用的方法需为 `public`，并加 `@JavascriptInterface` 注解。

R3.2 WebView：使用安全警告

警告：使用 `addJavascriptInterface` 可让 JavaScript 控制您的 Android 应用。虽然这可能很有用，但也可能会造成危险的安全问题。

如果 WebView 中的 HTML 不可信（例如，部分或全部 HTML 由未知人员或进程提供），则攻击者可以包含执行客户端代码的 HTML，并且可能包含攻击者选择的任何代码。

因此，除非您编写了在 WebView 中显示的所有 HTML 和 JavaScript，否则请不要使用 `addJavascriptInterface()`。请勿让用户在您的 WebView 中导航到不属于您自己的网页。而是让用户的默认浏览器应用打开外部链接。默认情况下，用户的网络浏览器会打开所有网址链接，因此此警告主要适用于您自行处理网页导航的情况。

R3.3 JSBridge：基本概念与使用场景

JSBridge（JavaScript 桥接）是 WebView 网页与宿主 Android 代码之间的通信机制：网页提出请求，应用执行开放的功能，并按约定返回结果。

- **接入原生交互**：网页触发应用提供的扫码、分享面板或生物识别流程，结果再传回网页。
- **复用应用逻辑与数据**：网页请求 Java / Kotlin 代码完成计算，或读取应用允许提供的配置。
- **同步界面与业务状态**：应用把任务完成、登录状态变化等结果通知网页，网页更新显示。

例：网页发起扫码

网页发送 `scan` 请求 → 应用检查权限并打开扫码界面 → 将扫码结果返回网页。

桥接约定包括：操作名称、参数格式、结果 / 错误，以及请求和回复如何对应。

R3.3 JSBridge：常用接口与安全风险

接口	主要用途	潜在安全风险
`addJavascriptInterface()`	网页调用 Java 对象方法	恶意页面或 iframe 可调用暴露的方法，触发应用功能。
`evaluateJavascript()`	应用执行页面脚本	拼接不可信数据时，可能注入并执行恶意脚本。
`WebViewCompat.addWebMessageListener()`	接收网页消息并回复	来源规则过宽时，恶意页面也可发送消息、请求操作。
`postWebMessage()` `WebMessagePort`	向页面发消息、建立通道	目标来源或端口交付不当，可能泄露消息或接收恶意请求。

- **恶意页面调用接口**：不可信页面、iframe 或脚本注入可能触发应用功能；限定来源，减少暴露能力。
- **接口越权与数据泄露**：校验操作、参数与业务权限，限制返回数据；来源规则避免使用 `*`。

R4 Jetpack Compose

Jetpack Compose 是 Android 构建原生界面的工具包：用 Kotlin 函数描述界面，状态变化时自动更新相关内容。

```
@Composable
fun Counter() {
  val count = remember {
    mutableIntStateOf(0)
  }
  Button(onClick = { count.intValue++ }) {
    Text("Clicked: ${count.intValue}")
  }
}
```

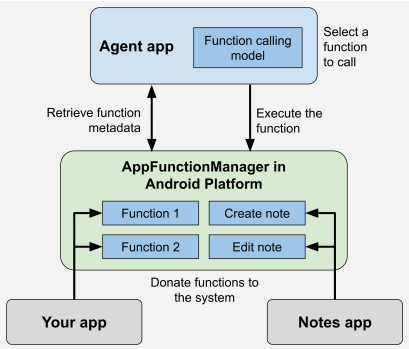
1. **描述界面**：`@Composable` 标记界面函数；`Button` 内放置 `Text`。
2. **保存状态**：`remember` 保留计数状态，初始值为 0。
3. **点击后更新**：点击让计数加 1，Compose 重新计算相关界面，按钮文字随之变化。

最终效果：按钮显示点击次数；这段界面无需编写布局 XML。

对安全分析的影响：沿界面函数、状态与回调追踪行为；状态更新未必触发 Activity 切换。

R5 AppFunction

AppFunctions 让 App 把“创建笔记”等功能声明为可调用的函数，供获授权的应用或 AI 助手发现并执行（Android 16）。



1. **声明功能**：App 描述函数的用途、参数和返回结果。
2. **登记与发现**：工具生成函数说明，由系统登记；获授权的调用方查询可用功能。
3. **选择与执行**：助手根据用户要求选择函数、传入参数，由目标 App 执行。

例如：用户说“创建一条购物笔记”，助手可调用笔记 App 的创建函数。

安全关注：检查调用权限、参数和业务授权。

开发基础与安全分析的关系

工程构建

知识点

工程文件；构建、调试与安装包。

安全分析里的作用

定位实现与权限，核对最终产物。

相关安全威胁

入口权限缺失导致越权调用。

核对导出设置与调用权限。

应用组件

知识点

组件、生命周期与界面管理。

安全分析里的作用

定位入口和回调，复现触发条件。

相关安全威胁

Service 越权、Provider 路径穿越。

限制调用者，校验真实路径。

数据流转

知识点

Intent、广播与 URI 授权。

安全分析里的作用

追踪接收者与数据共享范围。

相关安全威胁

隐式 Intent 泄露、粘性广播泄露或篡改。

限定接收方，最小授权；敏感数据不用粘性广播。

混合依赖

知识点

JNI、SDK 与 WebView；Compose 与应用函数。

安全分析里的作用

追踪跨语言调用、网页桥接与函数入口。

相关安全威胁

本地库 UAF、SDK 跟踪、网页越权调用。

检查内存与标识符；限制网页来源和接口。