一、Activity

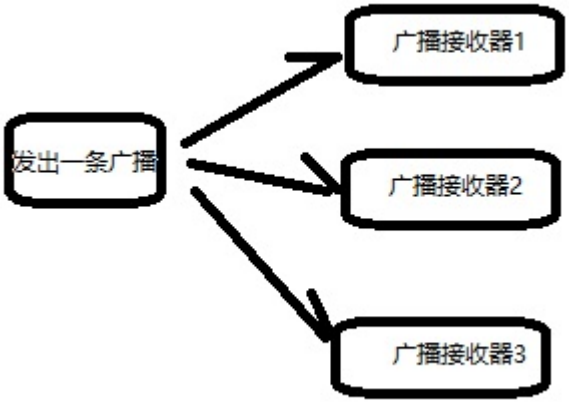
二、Brodcast

Android提供的一套完整的API，允许应用程序发送和接受系统或其他应用的广播消息。感觉就是进程间通信的一种机制。

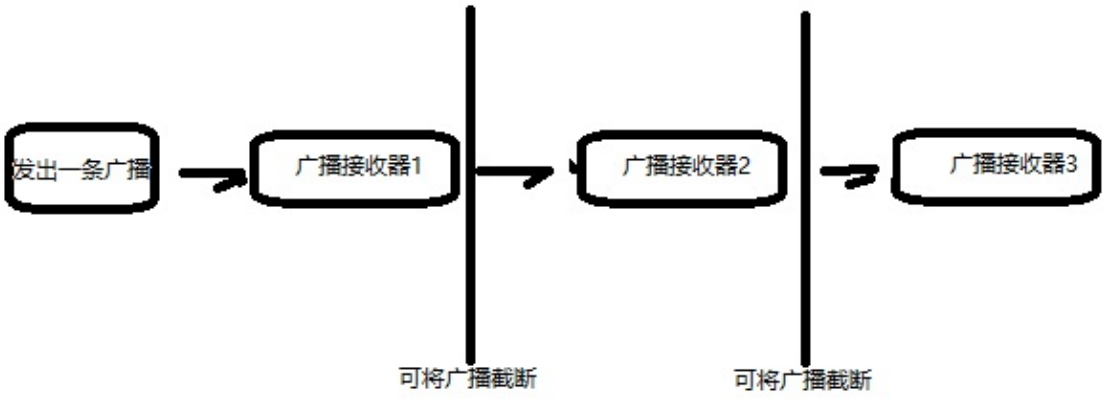
每个应用程序自己对感兴趣的时间进行注册订阅。

Android中的广播分为两种类型：

- **标准广播**:是一种异步广播一条广播发出后，所有的广播接收器几乎会在同一时间接收到这条广播消息，他们之间不存在先后关系或优先级顺序。所以无法被截断。



- **有序广播**：是一种同步广播，广播发出后，同一时刻只有一个广播接收器会受到该条广播消息，当这个广播接收器中的逻辑执行完成后，在传递给下一个。
  广播接收器有先后顺序，优先级高的先收到，并且可以截断消息。



两种注册方式：

- **动态注册**：代码中注册。要求程序运行才能起作用。优点在于：可以自由控制注册与注销，灵活；缺点是：程序必须启动才能接收广播，因为注册逻辑写在onCreat()方法中。比方开机广播，就接受不到。
- **静态注册**：在Androidmanifest中注册。

接受系统广播

Android内置系统广播：包括不限于

- 手机开机
- 电量变化
- 时间或区域发生改变

动态注册监听网络变化

> **接收器的创建方式：**
>
> 1. 创建类，继承BroadcastReceiver
> 2. 重写父类onRecive9()方法，在该方法中写处理逻辑。
> 3. 在活动的onCreat()方法中创建intentFilter，addAction，然后将继承自BroadcastReceiver类的实例和intentFilter一起作为参数注册（逻辑较为简单，intntFilter指明要听什么消息，自己写的类写明接到消息后干什么。）
>   （有些类似于intent启动其他活动的用法，当时intent的银式使用课题在activity中配置自己可以响应什么事件，通过标签，动态的intent是创建要产生的事件）

```
public class MainActivity extends AppCompatActivity {
    private IntentFilter intentFilter;
    private NetWorkChangeReceiver receiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        intentFilter = new IntentFilter();
        intentFilter.addAction("android.net.conn.CONNECTIVITY_CHANGE");
        receiver = new NetWorkChangeReceiver();
        registerReceiver(receiver, intentFilter);
    }

    class NetWorkChangeReceiver extends BroadcastReceiver{
        @Override
        public void onReceive(Context context, Intent intent) {
            ConnectivityManager connectivityManager = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE)
            NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo(); // 这里需要用户权限，要在配置文件中配置
            if (networkInfo != null && networkInfo.isAvailable()){
                Toast.makeText(context, "network is available", Toast.LENGTH_SHORT).show();
            }else{
                Toast.makeText(context, "network  is uiavilable", Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

在AndroidManifest.xml中加权限

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

静态注册实现开机启动

> **创建方式：**
>
> 1. new -> other -> BroadcastReceiver
> 2. 重写onReceive
> 3. 在AndroidManifest.xml中标签中注册

发送自定义广播

（广播其实就是给个action，通过这个action来区别的，就像每个消息有个key，上面的"android.net.conn.CONNECTIVITY_CHANGE"就是一种key，所以自定义就是自定义一个action的key）

发送标准广播

> **创建方式**
>
> 1. 先创建一个接受者，无论动态还是静态，都一定会有

```
IntentFilter.addAction()
```

> 或者

```
<intent - filter>
 <action ...>
</intent - filter>
```

> 这里面会指定一个action，就是一个字符串(很关键)
>
> 2. new 一个Intent，通过构造函数传进action的名字（那个字符串），然后send(intent) ,intent可以传递数据。

接收不到解决：

- 加参数：

```
intent.setComponent(new ComponentName("com.example.broadcaststudy",
                      "com.example.broadcaststudy.MyBroadcastReciver"));
```

- 用动态代替静态

发送有序广播

把sendBroadcast()方法换成sendOrderedBroadcast(intent, <权限相关字符串，传null可行>)即可。在intent-filter中可以设置优先级，在onReceive()方法中可以拦截丢弃。

使用本地广播

本地广播对比系统全局广播：

- 系统全局广播可以被其他任何应用程序接收到，并且可以接受来自其他任何应用的广播，因此容易引起安全性问题
- 本地广播发出的广播只会在应用程序内部传递，并且只接受来自本应用的广播。
- 本地广播无法通过静态注册的方式实现。
- 本地广播较高效

三、Providers

> 跨程序共享数据。提供了完整机制，允许一个程序访问另一个程序中的数据，同时还能保证数据的安全性。

> 与SharedPreferences相比:不同于SPf中的两种全局可读可写操作模式，内容提供器可以选择对哪一部分数据进行共享，从而保证隐私数据不会泄露

运行时权限

在Androidmanifest中加入后，低于Android 6.0的手机在安装时，会提示相关权限申请，拒绝则不安装，不决绝则安装，所以存在滥用权限的情况。

Android 6.0之后加入运行时权限，在运行过程中申请，拒绝也不会影响其他功能的使用。

Android将所有权限分为两类：

- 普通权限：不会直接威胁到用户的安全和隐私的权限，系统会自动进行授权
- 危险权限：可能会触及用户隐私或者对设备安全性造成影响的权限，如获取联系人信息，定位设备的地理位置等，这部分必须由用户自己手动点击授权才可以。

危险权限：

权限组名	权限名
CaLENDAR	READ_CALENDAR 、WRITE_CALENDAR
CAMEAR	CAMERA
CONTACTS	READ_CONTECTS 、WRITE_CONTACTS
…	…

(用到时，查表)在进行权限处理时，使用的是权限名，但是用户一旦同意授权了，那么该权限所对应的权限组中所有的其他权限也会同时被授权。

运行时申请权限

还是通过Intent，所以总体来说，在活动或者界面之间穿来穿去，都可以用intent:

- 活动之间切换用显式intent
- broadcast也是活动之间或者进程之间通信，也必须用IntentFilter去作为接收的参数，发送广播也要用send(intent)

Intent里面比较重要的就是构造函数中的Action参数，指定传输的目的地！Intent还可以当做数据容器传递数据。

访问其他程序中的数据

> 使用方法有两种：
>
> - 使用现有的内容提供器来读取和操作相应程序中的数据
> - 创建自己的内容提供器给程序的数据提供外部访问接口

如果一个应用程序通过内容提供器对其数据提供了外部访问接口，那么任何其他应用程序就都可以使用这部分数据进行访问。Android中自带的电话薄、短信、媒体等程序都提供了类似的访问接口，这就使得第三方应用程序可以充分地利用这部分数据来实现更好的功能。

ContentResolver的基本用法

如果一个应用程序想访问内容提供器中共享的数据，就一定要借助ContentResolver类。

ContentResolver提供了一系列方法用于对数据进行CRUD操作，ContentResolver中的增删改查方法**不接收表名**，而是**用Uri代替**，该参数称为URI，对数据建立唯一标识符，主要有两部分构成：

- authority : 用于区分不同程序
- path ： 对同一应用程序中的不同表做区分，通常会添加到authority后面。

URI内容最标准的写法<协议名: authority path>：

```
content://com.example.app.provider/table1
```

在得到内容URI字符串后，还需要将他解析成Uri对象才能传入，解析方法为：

```
Uri uri = Uri.parse("content://com.example.app.provider/table1");
```

然后查询：

```
Cursor cursor = getContentResolver().query(
uri,
projection,
selection,
selectionArgs,
sortOrder);
```

和SQLiteDatabase中的query方法对比：

query()方法参数	对应SQL部分	描述
uri	from table_name	指定查询某个应用程序下的某一张表
projection	select column1, column2	指定查询的列名
selection	where column = value	指定where的约束条件
selectionArgs	-	为where重的占位符提供具体的值
orderBy	order by column1, column2	指定查询结果的排序方式

查询完成后返回的时一个Cursor对象，逐个读取：

```
if(cursor != null){
    while(cursor.moveToNext()){
        String column1 = cursor.getString(cursor.getColumnIndex("column1"));
        int column2 = cursor.getInt(cursor.getColumnIndex("column2"));
    }
}
```

增加数据：

```
ContentValues values = new ContentValues();
values.put("column1", "text");
values.put("column2", 1);
getContentResolver().insert(uri, values);
```

更新数据：

```
ContentValues values = new ContentValues();
values.put("column1", "");
getContentResolver().update(uri,
                            values,
```

```
                            "column1 = ? and colmun2 = ?",
                            new String[]{"text", "1"});
```

删除：

```
getContentResolver().delete(uri, "column1 = ? , new String[]{"text"});
```

创建自己的内容提供器：

创建内容提供器的步骤

方式：新建一个类去继承ContentProvider，实现里面的方法，就是对外提供的接口实现。

除了标准的URI之外，还要一种后面带一个id

```
content://com.example.app.provider/table1/1
```

表示期望访问的是com.example.app.provider这哥应用的table1中id为1的数据。**以路径结尾表示要访问表中的所有数据，以id访问表示期望表中拥有相应id的数据**，可以使用通配符的方式来分配这两种格式内容URI:

- *:表示匹配任意长度的任意字符
- #：表示匹配任意长度的数字

一个能够匹配任意表内容的URI:

```
content://com.example.app.provider/*
```

一个能够匹配table1表中任意一行数据内容URI：

```
content://com.example.app.provider/table1/#
```

可以借助UriMatch这个类轻松匹配内容URI的功能。

```
public class MyProvider extends ContentProvider {

    public static final int TABLE1_DIR = 0;
    public static final int TABLE1_ITEM = 1;
    public static final int TABLE2_DIR = 2;
    public static final int TABLE2_ITEM = 3;

    private static UriMatcher uriMatcher;

    static {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI("com.example.runtimepermissiontest.provider",
                "table1", TABLE1_DIR);
        uriMatcher.addURI("com.example.runtimepermissiontest.provider",
                "table1", TABLE1_ITEM);
        uriMatcher.addURI("com.example.runtimepermissiontest.provider",
                "table2", TABLE2_DIR);
        uriMatcher.addURI("com.example.runtimepermissiontest.provider",
                "table2", TABLE2_ITEM);
    }

    /**
```

```
     * 当ContentResolver尝试访问当前程序数据时，内容提供器被初始化，此时会调用该函数
     * 通常在这里完成度数据库的创建和升级操作
     * 返回true表示内容提供器初始化成功，返回false表示失败
     */
    @Override
    public boolean onCreate() {
        return false;
    }


    @Override
    public Cursor query( Uri uri,  String[] projection, String selection,  String[] selectionArgs,  String sortOrder) {
        switch (uriMatcher.match(uri)){
            case TABLE1_DIR:
                // 查询table1中的内容
                break;
            case TABLE1_ITEM:
                //查询table1中的单条内容
                break;
            case TABLE2_DIR:
                // 查询table2中的内容
                break;
            case TABLE2_ITEM:
                // 查询table2中的单条内容
                break;
            default:
                break;
        }
        return null;
    }

    /**
     * 根据传入的内容URI来返回相应的MIME类型
     * 是一种固定格式
     */
    @Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)){
            case TABLE1_DIR:
                return "vnd.android.cursor.dir/vnd.com.example.runtimepermissiontest.provider.table1";
            case TABLE1_ITEM:
                return "vnd.android.cursor.item/vnd.com.example.runtimepermissiontest.provider.table1";
            case TABLE2_DIR:
                return "vnd.android.cursor.dir/vnd.com.example.runtimepermissiontest.provider.table2";
            case TABLE2_ITEM:
                return "vnd.android.cursor.item/vnd.com.example.runtimepermissiontest.provider.table2";
            default:
                break;
        }
        return null;
    }

    @Override
    public Uri insert( Uri uri,  ContentValues values) {
        return null;
    }

    @Override
    public int delete( Uri uri,  String selection,  String[] selectionArgs) {
        return 0;
    }

    @Override
    public int update( Uri uri, ContentValues values,  String selection, String[] selectionArgs) {
        return 0;
    }
}
```

> 关于getTypoe()方法用于获取Uri对象所对应的MIME类型：
>
> - 一个内容URI所对应的MIME字符串主要由3部分组成

> - 必须以 vnd 开头
> - 如果内容URI以路径结尾，则后接android.cursor.dir/，如果内容URI以id结尾，则后接android.cursor.item/。
> - 最后接上 vnd.<authority>.<path>

四、Service