

第 5 章 全局大喇叭，详解广播机制

记得在我上学的时候，每个班级的教室里都会装有一个喇叭，这些喇叭都是接入到学校的广播室的，一旦有什么重要的通知，就会播放一条广播来告知全校的师生。类似的工作机制其实在计算机领域也有很广泛的应用，如果你了解网络通信原理应该会知道，在一个 IP 网络范围中最大的 IP 地址是被保留作为广播地址来使用的。比如某个网络的 IP 范围是 192.168.0.XXX，子网掩码是 255.255.255.0，那么这个网络的广播地址就是 192.168.0.255。广播数据包会被发送到同一网络上的所有端口，这样在该网络中的每台主机都将会收到这条广播。

为了方便于进行系统级别的消息通知，Android 也引入了一套类似的广播消息机制。相比于我前面举出的两个例子，Android 中的广播机制会显得更加的灵活，本章就将对这一机制的方方面面进行详细的讲解。

5.1 广播机制简介

为什么说 Android 中的广播机制更加灵活呢？这是因为 Android 中的每个应用程序都可以对自己感兴趣的广播进行注册，这样该程序就只会接收到自己所关心的广播内容，这些广播可能是来自于系统的，也可能是来自于其他应用程序的。Android 提供了一套完整的 API，允许应用程序自由地发送和接收广播。发送广播的方法其实之前稍微有提到过一下，如果你记性好的话可能还会有印象，就是借助我们第 2 章学过的 Intent。而接收广播的方法则需要引入一个新的概念，广播接收器（Broadcast Receiver）。

广播接收器的具体用法将会在下一节中做介绍，这里我们先来了解一下广播的类型。Android 中的广播主要可以分为两种类型，标准广播和有序广播。

标准广播（Normal broadcasts）是一种完全异步执行的广播，在广播发出之后，所有的广播接收器几乎都会在同一时刻接收到这条广播消息，因此它们之间没有任何先后顺序可言。这种广播的效率会比较高，但同时也意味着它是无法被截断的。标准广播的工作流程如图 5.1 所示。

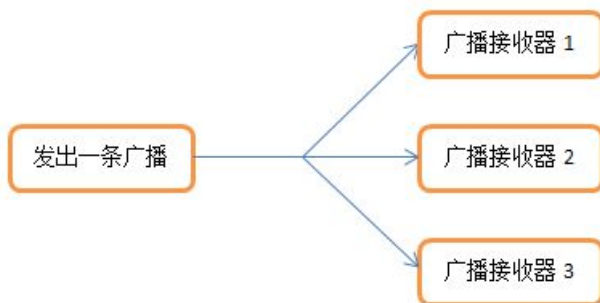


图 5.1

有序广播（Ordered broadcasts）则是一种同步执行的广播，在广播发出之后，同一时刻只会有一个广播接收器能够收到这条广播消息，当这个广播接收器中的逻辑执行完毕后，广播才会继续传递。所以此时的广播接收器是有先后顺序的，优先级高的广播接收器就可以先收到广播消息，并且前面的广播接收器还可以截断正在传递的广播，这样后面的广播接收器就无法收到广播消息了。有序广播的工作流程如图 5.2 所示。

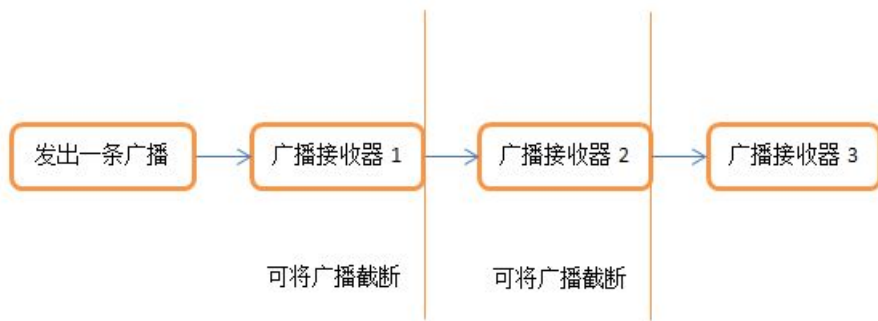


图 5.2

掌握了这些基本概念后，我们就可以来尝试一下广播的用法了，首先就从接收系统广播开始吧。

5.2 接收系统广播

Android 内置了很多系统级别的广播，我们可以在应用程序中通过监听这些广播来得到各种系统的状态信息。比如手机开机完成后会发出一条广播，电池的电量发生变化会发出一条广播，时间或时区发生改变也会发出一条广播等等。如果想要接收到这些广播，就需要使用广播接收器，下面我们就来看一下它的具体用法。

5.2.1 动态注册监听网络变化

广播接收器可以自由地对自己感兴趣的广播进行注册，这样当有相应的广播发出时，广播接收器就能够收到该广播，并在内部处理相应的逻辑。注册广播的方式一般有两种，在代码中注册和在 AndroidManifest.xml 中注册，其中前者也被称为动态注册，后者也被称为静态注册。

那么该如何创建一个广播接收器呢？其实只需要新建一个类，让它继承自 BroadcastReceiver，并重写父类的 onReceive() 方法就行了。这样当有广播到来时，onReceive() 方法就会得到执行，具体的逻辑就可以在这个方法中处理。

那我们就先通过动态注册的方式编写一个能够监听网络变化的程序，借此学习一下广播接收器的基本用法吧。新建一个 BroadcastTest 项目，然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private IntentFilter intentFilter;

    private NetworkChangeReceiver networkChangeReceiver;

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

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(networkChangeReceiver);
    }

    class NetworkChangeReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
```

```

        Toast.makeText(context, "network changes",
            Toast.LENGTH_SHORT).show();
    }

}

```

可以看到，我们在 MainActivity 中定义了一个内部类 NetworkChangeReceiver，这个类是继承自 BroadcastReceiver 的，并重写了父类的 onReceive() 方法。这样每当网络状态发生变化时，onReceive() 方法就会得到执行，这里只是简单地使用 Toast 提示了一段文本信息。

然后观察 onCreate() 方法，首先我们创建了一个 IntentFilter 的实例，并给它添加了一个值为 android.net.conn.CONNECTIVITY_CHANGE 的 action，为什么要添加这个值呢？因为当网络状态发生变化时，系统发出的正是一条值为 android.net.conn.CONNECTIVITY_CHANGE 的广播，也就是说我们的广播接收器想要监听什么广播，就在这里添加相应的 action 就行了。接下来创建了一个 NetworkChangeReceiver 的实例，然后调用 registerReceiver() 方法进行注册，将 NetworkChangeReceiver 的实例和 IntentFilter 的实例都传了进去，这样 NetworkChangeReceiver 就会收到所有值为 android.net.conn.CONNECTIVITY_CHANGE 的广播，也就实现了监听网络变化的功能。

最后要记得，动态注册的广播接收器一定都要取消注册才行，这里我们是在 onDestroy() 方法中通过调用 unregisterReceiver() 方法来实现的。

整体来说，代码还是非常简单的，现在运行一下程序。首先你会在注册完成的时候收到一条广播，然后按下 Home 键回到主界面（注意不能按 Back 键，否则 onDestroy() 方法会执行），接着按下 Menu 键→System settings→Data usage 进入到数据使用详情界面，然后尝试着开关 Mobile Data 来启动和禁用网络，你就会看到有 Toast 提醒你网络发生了变化。

不过只是提醒网络发生了变化还不够人性化，最好是能准确地告诉用户当前是有网络还是没有网络，因此我们还需要对上面的代码进行进一步的优化。修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends Activity {
    .....

    class NetworkChangeReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            ConnectivityManager connectionManager = (ConnectivityManager)
                getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo networkInfo = connectionManager.getActiveNetworkInfo();

```

```

        if (networkInfo != null && networkInfo.isAvailable()) {
            Toast.makeText(context, "network is available",
                Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(context, "network is unavailable",
                Toast.LENGTH_SHORT).show();
        }
    }
}
}

```

在 `onReceive()` 方法中，首先通过 `getSystemService()` 方法得到了 `ConnectivityManager` 的实例，这是一个系统服务类，专门用于管理网络连接的。然后调用它的 `getActiveNetworkInfo()` 方法可以得到 `NetworkInfo` 的实例，接着调用 `NetworkInfo` 的 `isAvailable()` 方法，就可以判断出当前是否有网络了，最后我们还是通过 `Toast` 的方式对用户进行提示。

另外，这里有非常重要的一点需要说明，Android 系统为了保证应用程序的安全性做了规定，如果程序需要访问一些系统的关键性信息，必须在配置文件中声明权限才可以，否则程序将会直接崩溃，比如这里查询系统的网络状态就是需要声明权限的。打开 `AndroidManifest.xml` 文件，在里面加入如下权限就可以查询系统网络状态了：

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="19" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    .....
</manifest>

```

访问 <http://developer.android.com/reference/android/Manifest.permission.html> 可以查看 Android 系统所有可声明的权限。

现在重新运行程序，然后按下 Home 键→按下 Menu 键→System settings→Data usage 进入到数据使用详情界面，关闭 Mobile Data 会弹出无网络可用的提示，如图 5.3 所示。

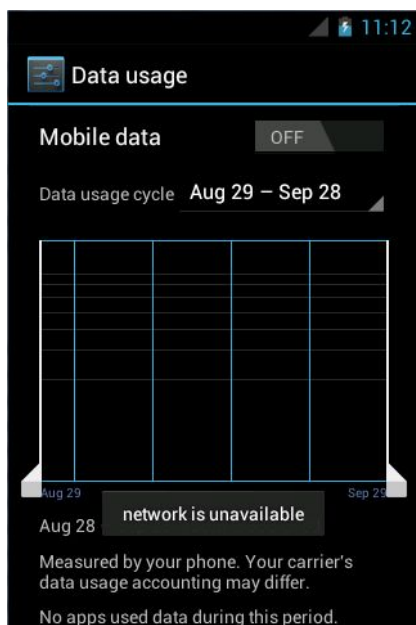


图 5.3

然后重新打开 Mobile Data 又会弹出网络可用的提示，如图 5.4 所示。

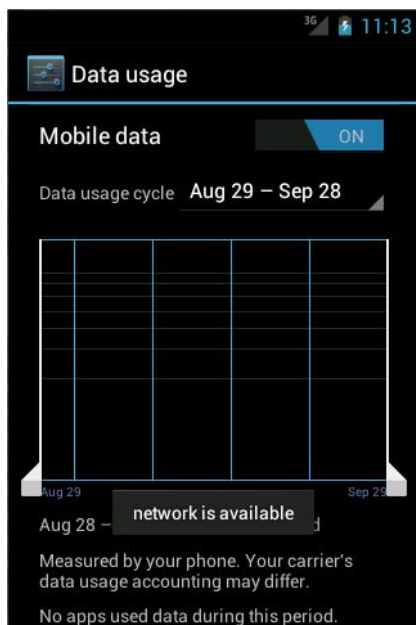


图 5.4

5.2.2 静态注册实现开机启动

动态注册的广播接收器可以自由地控制注册与注销，在灵活性方面有很大的优势，但是它也存在着一个缺点，即必须要在程序启动之后才能接收到广播，因为注册的逻辑是写在 `onCreate()` 方法中的。那么有没有什么办法可以让程序在未启动的情况下就能接收到广播呢？这就需要使用静态注册的方式了。

这里我们准备让程序接收一条开机广播，当收到这条广播时就可以在 `onReceive()` 方法里执行相应的逻辑，从而实现开机启动的功能。新建一个 `BootCompleteReceiver` 继承自 `BroadcastReceiver`，代码如下所示：

```
public class BootCompleteReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Boot Complete", Toast.LENGTH_LONG).show();
    }

}
```

可以看到，这里不再使用内部类的方式来定义广播接收器，因为稍后我们需要在 `AndroidManifest.xml` 中将这个广播接收器的类名注册进去。在 `onReceive()` 方法中，还是简单地使用 `Toast` 弹出一段提示信息。

然后修改 `AndroidManifest.xml` 文件，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest"
    android:versionCode="1"
    android:versionName="1.0" >
    .....

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        .....

        <receiver android:name=".BootCompleteReceiver" >
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
```

```

        </intent-filter>
    </receiver>
</application>
</manifest>

```

终于，<application>标签内出现了一个新的标签<receiver>，所有静态注册的广播接收器都是在这里进行注册的。它的用法其实和<activity>标签非常相似，首先通过 `android:name` 来指定具体注册哪一个广播接收器，然后在<intent-filter>标签里加入想要接收的广播就行了，由于 Android 系统启动完成后会发出一条值为 `android.intent.action.BOOT_COMPLETED` 的广播，因此我们在这里添加了相应的 `action`。

另外，监听系统开机广播也是需要声明权限的，可以看到，我们使用<uses-permission>标签又加入了一条 `android.permission.RECEIVE_BOOT_COMPLETED` 权限。

现在重新运行程序后，我们的程序就已经可以接收开机广播了，首先打开到应用程序管理界面来查看一下当前程序所拥有的权限。在桌面按下 Menu 键→System settings→Apps，然后点击 BroadcastTest，如图 5.5 所示。

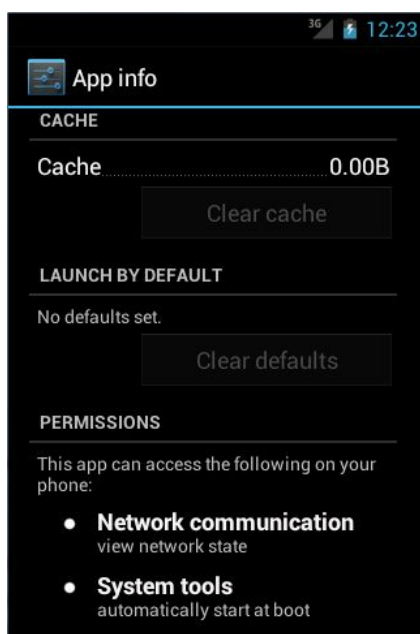


图 5.5

可以看到，我们的程序目前拥有访问网络状态和开机自动启动的权限。然后将模拟器关闭并重新启动，在启动完成之后就会收到开机广播了，如图 5.6 所示。

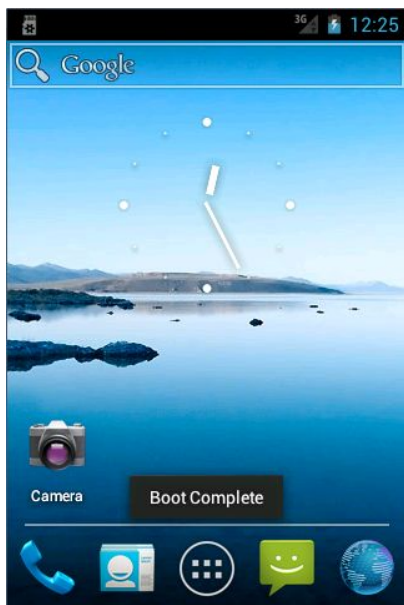


图 5.6

到目前为止，我们在广播接收器的 `onReceive()` 方法中都只是简单地使用 `Toast` 提示了一段文本信息，当你真正在项目中使用到它的时候，就可以在里面编写自己的逻辑。需要注意的是，不要在 `onReceive()` 方法中添加过多的逻辑或者进行任何的耗时操作，因为在广播接收器中是不允许开启线程的，当 `onReceive()` 方法运行了较长时间而没有结束时，程序就会报错。因此广播接收器更多的是扮演一种打开程序其他组件的角色，比如创建一条状态栏通知，或者启动一个服务等，这几个概念我们会在后面的章节中学到。

经验值：+3000

目前经验值：19905

级别：鸟

赢得宝物：战胜广播尊者。拾取广播尊者掉落的宝物，一台小的调频收音机，还有一个在神界乡村常用的那种大喇叭头子，主要用来向那些乡下神广播神界的惠农政策。说到乡下神，乡下牛是他们的好伙伴，乡下牛并不是乡下神养殖的家畜，他们之间完全是平等的合作关系，同属神族公民，农忙时互相配合。广播尊者虽身为尊者，但特别喜欢干农活，一天到晚活跃在田间地头，为乡下神带去最新的农业技术，以及城里神的新鲜事。

5.3 发送自定义广播

现在你已经学会了通过广播接收器来接收系统广播，接下来我们就要学习一下如何在应

用程序中发送自定义的广播。前面已经介绍过了，广播主要分为两种类型，标准广播和有序广播，在本节中我们就将通过实践的方式来看下这两种广播具体的区别。

5.3.1 发送标准广播

在发送广播之前，我们还是需要先定义一个广播接收器来准备接收此广播才行，不然发出去也是白发。因此新建一个 MyBroadcastReceiver 继承自 BroadcastReceiver，代码如下所示：

```
public class MyBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "received in MyBroadcastReceiver",
            Toast.LENGTH_SHORT).show();
    }

}
```

这里当 MyBroadcastReceiver 收到自定义的广播时，就会弹出 received in MyBroadcastReceiver 的提示。然后在 AndroidManifest.xml 中对这个广播接收器进行注册：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest"
    android:versionCode="1"
    android:versionName="1.0" >
    .....
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        .....
        <receiver android:name=".MyBroadcastReceiver">
            <intent-filter>
                <action android:name="com.example.broadcasttest.MY_BROADCAST"/>
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

可以看到，这里让 MyBroadcastReceiver 接收一条值为 com.example.broadcasttest.MY_BROADCAST 的广播，因此待会儿在发送广播的时候，我们就需要发出这样的一条广播。

接下来修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send Broadcast"
    />

</LinearLayout>
```

这里在布局文件中定义了一个按钮，用于作为发送广播的触发点。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {
    .....

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent("com.example.broadcasttest.
MY_BROADCAST");
                sendBroadcast(intent);
            }
        });
        .....
    }
    .....
}
```

可以看到，我们在按钮的点击事件里面加入了发送自定义广播的逻辑。首先构建出了一个 Intent 对象，并把要发送的广播的值传入，然后调用了 Context 的 sendBroadcast() 方法将广播发送出去，这样所有监听 com.example.broadcasttest.MY_BROADCAST 这条广播的广播接收器就会收到消息。此时发出去的广播就是一条标准广播。

重新运行程序，并点击一下 Send Broadcast 按钮，效果如图 5.7 所示。

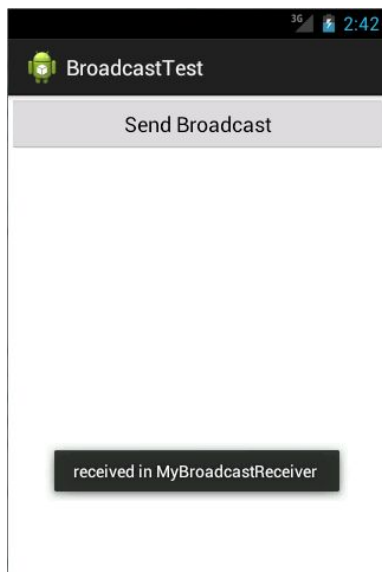


图 5.7

这样我们就成功完成了发送自定义广播的功能。另外，由于广播是使用 Intent 进行传递的，因此你还可以在 Intent 中携带一些数据传递给广播接收器。

5.3.2 发送有序广播

广播是一种可以跨进程的通信方式，这一点从前面接收系统广播的时候就可以看出来。因此在我们应用程序内发出的广播，其他的应用程序应该也是可以收到的。为了验证这一点，我们需要再新建一个 BroadcastTest2 项目。

将项目创建好之后，还需要在这个项目下定义一个广播接收器，用于接收上一小节中的自定义广播。新建 AnotherBroadcastReceiver 继承自 BroadcastReceiver，代码如下所示：

```
public class AnotherBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "received in AnotherBroadcastReceiver",
            Toast.LENGTH_SHORT).show();
    }

}
```

这里仍然是在广播接收器的 `onReceive()` 方法中弹出了一段文本信息。然后在 `AndroidManifest.xml` 中对这个广播接收器进行注册，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest2"
    android:versionCode="1"
    android:versionName="1.0" >
    .....
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        .....
        <receiver android:name=".AnotherBroadcastReceiver" >
            <intent-filter>
                <action android:name="com.example.broadcasttest.MY_BROADCAST" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

可以看到，`AnotherBroadcastReceiver` 同样接收的是 `com.example.broadcasttest.MY_BROADCAST` 这条广播。现在运行 `BroadcastTest2` 项目将这个程序安装到模拟器上，然后重新回到 `BroadcastTest` 项目的主界面，并点击一下 `Send Broadcast` 按钮，就会分别弹出两次提示信息，如图 5.8 所示。

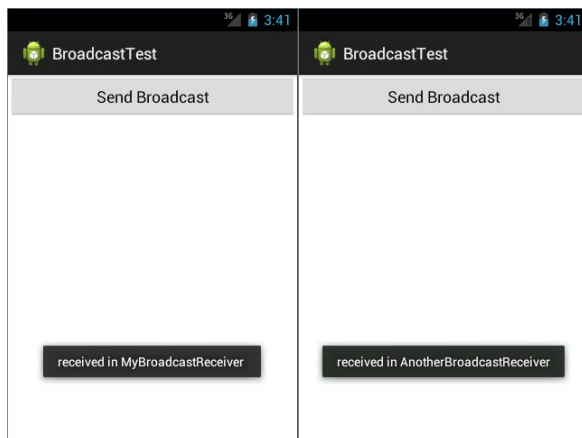


图 5.8

这样就强有力地证明了，我们的应用程序发出的广播是可以被其他的应用程序接收到的。不过到目前为止，程序里发出的都还是标准广播，现在我们来尝试一下发送有序广播。关闭 BroadcastTest2 项目，然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {
    .....

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent("com.example.broadcasttest.
MY_BROADCAST");
                sendOrderedBroadcast(intent, null);
            }
        });
        .....
    }
    .....
}
```

可以看到，发送有序广播只需要改动一行代码，即将 sendBroadcast() 方法改成 sendOrderedBroadcast() 方法。sendOrderedBroadcast() 方法接收两个参数，第一个参数仍然是 Intent，第二个参数是一个与权限相关的字符串，这里传入 null 就行了。现在重新运行程序，并点击 Send Broadcast 按钮，你会发现，两个应用程序仍然都可以接收到这条广播。

看上去好像和标准广播没什么区别嘛，不过别忘了，这个时候的广播接收器是有先后顺序的，而且前面的广播接收器还可以将广播截断，以阻止其继续传播。

那么该如何设定广播接收器的先后顺序呢？当然是在注册的时候进行设定的了，修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest"
    android:versionCode="1"
    android:versionName="1.0" >
    .....
    <application
        android:allowBackup="true"
```

```

        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        .....
        <receiver android:name=".MyBroadcastReceiver">
            <intent-filter android:priority="100" >
                <action android:name="com.example.broadcasttest.MY_BROADCAST"/>
            </intent-filter>
        </receiver>
    </application>
</manifest>

```

可以看到，我们通过 `android:priority` 属性给广播接收器设置了优先级，优先级比较高的广播接收器就可以先收到广播。这里将 `MyBroadcastReceiver` 的优先级设成了 100，以保证它一定会在 `AnotherBroadcastReceiver` 之前收到广播。

既然已经获得了接收广播的优先权，那么 `MyBroadcastReceiver` 就可以选择是否允许广播继续传递了。修改 `MyBroadcastReceiver` 中的代码，如下所示：

```

public class MyBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "received in MyBroadcastReceive",
            Toast.LENGTH_SHORT).show();
        abortBroadcast();
    }

}

```

如果在 `onReceive()` 方法中调用了 `abortBroadcast()` 方法，就表示将这条广播截断，后面的广播接收器将无法再接收到这条广播。现在重新运行程序，并点击一下 `Send Broadcast` 按钮，你会发现，只有 `MyBroadcastReceiver` 中的 `Toast` 信息能够弹出，说明这条广播经过 `MyBroadcastReceiver` 之后确实是终止传递了。

5.4 使用本地广播

前面我们发送和接收的广播全部都是属于系统全局广播，即发出的广播可以被其他任何的任何应用程序接收到，并且我们也可以接收来自于其他任何应用程序的广播。这样就很容易会引起安全性的问题，比如说我们发送的一些携带关键性数据的广播有可能被其他的应用

程序截获，或者其他的程序不停地向我们的广播接收器里发送各种垃圾广播。

为了能够简单地解决广播的安全性问题，Android 引入了一套本地广播机制，使用这个机制发出的广播只能够在应用程序的内部进行传递，并且广播接收器也只能接收来自本应用程序发出的广播，这样所有的安全性问题就都不存在了。

本地广播的用法并不复杂，主要就是使用了一个 LocalBroadcastManager 来对广播进行管理，并提供了发送广播和注册广播接收器的方法。下面我们就通过具体的实例来尝试一下它的用法，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private IntentFilter intentFilter;

    private LocalReceiver localReceiver;

    private LocalBroadcastManager localBroadcastManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        localBroadcastManager = LocalBroadcastManager.getInstance(this);
// 获取实例
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent("com.example.broadcasttest.
LOCAL_BROADCAST");
                localBroadcastManager.sendBroadcast(intent); // 发送本地广播
            }
        });
        intentFilter = new IntentFilter();
        intentFilter.addAction("com.example.broadcasttest.LOCAL_BROADCAST");
        localReceiver = new LocalReceiver();
        localBroadcastManager.registerReceiver(localReceiver, intentFilter);
// 注册本地广播监听器
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
    }
}
```



```
        localBroadcastManager.unregisterReceiver(localReceiver);  
    }  
  
    class LocalReceiver extends BroadcastReceiver {  
  
        @Override  
        public void onReceive(Context context, Intent intent) {  
            Toast.makeText(context, "received local broadcast",  
                Toast.LENGTH_SHORT).show();  
        }  
    }  
}
```

有没有感觉这些代码很熟悉？没错，其实这基本上就和我们前面所学的动态注册广播接收器以及发送广播的代码是一样。只不过现在首先是通过 LocalBroadcastManager 的 getInstance() 方法得到了它的一个实例，然后在注册广播接收器的时候调用的是 LocalBroadcastManager 的 registerReceiver() 方法，在发送广播的时候调用的是 LocalBroadcastManager 的 sendBroadcast() 方法，仅此而已。这里我们在按钮的点击事件里面发出了一条 com.example.broadcasttest.LOCAL_BROADCAST 广播，然后在 LocalReceiver 里去接收这条广播。重新运行程序，并点击 Send Broadcast 按钮，效果如图 5.9 所示。



图 5.9

可以看到，LocalReceiver 成功接收到了这条本地广播，并通过 Toast 提示了出来。如果你还有兴趣进行实验，可以尝试在 BroadcastTest2 中也去接收 com.example.broadcasttest.LOCAL_BROADCAST 这条广播，答案是显而易见的，肯定无法收到，因为这条广播只会在 BroadcastTest 程序内传播。

另外还有一点需要说明，本地广播是无法通过静态注册的方式来接收的。其实这也完全可以理解，因为静态注册主要就是为了让程序在未启动的情况下也能收到广播，而发送本地广播时，我们的程序肯定是已经启动了，因此也完全不需要使用静态注册的功能。

最后我们再来盘点一下使用本地广播的几点优势吧。

1. 可以明确地知道正在发送的广播不会离开我们的程序，因此不需要担心机密数据泄漏的问题。
2. 其他的程序无法将广播发送到我们程序的内部，因此不需要担心会有安全漏洞的隐患。
3. 发送本地广播比起发送系统全局广播将会更加高效。

5.5 广播的最佳实践——实现强制下线功能

本章的内容不是非常多，因此相信你也一定学得很轻松吧。现在我们就准备通过一个完整例子的实践，来综合运用一下本章中所学到的知识。

强制下线功能应该算是比较常见的了，很多的应用程序都具备这个功能，比如你的 QQ 号在别处登录了，就会将你强制挤下线。其实实现强制下线功能的思路也比较简单，只需要在界面上弹出一个对话框，让用户无法进行任何其他操作，必须要点击对话框中的确定按钮，然后回到登录界面即可。可是这样就存在着一个问题，因为我们被通知需要强制下线时可能正处于任何一个界面，难道需要在每个界面上都编写一个弹出对话框的逻辑？如果你真的这么想，那思维就偏远了，我们完全可以借助本章中所学的广播知识，来非常轻松地实现这一功能。新建一个 BroadcastBestPractice 项目，然后开始动手吧。

强制下线功能需要先关闭掉所有的活动，然后回到登录界面。如果你的反应足够快的话，应该会想到我们在第 2 章的最佳实践部分早就已经实现过关闭所有活动的功能了，因此这里只需要使用同样的方案即可。先创建一个 ActivityCollector 类用于管理所有的活动，代码如下所示：

```
public class ActivityCollector {  
  
    public static List<Activity> activities = new ArrayList<Activity>();  
  
    public static void addActivity(Activity activity) {  
        activities.add(activity);  
    }  
}
```

```

    }

    public static void removeActivity(Activity activity) {
        activities.remove(activity);
    }

    public static void finishAll() {
        for (Activity activity : activities) {
            if (!activity.isFinishing()) {
                activity.finish();
            }
        }
    }
}

```

然后创建 BaseActivity 类作为所有活动的父类，代码如下所示：

```

public class BaseActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityCollector.addActivity(this);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        ActivityCollector.removeActivity(this);
    }

}

```

接着需要创建一个登录界面的布局，还记得我们在 3.3.4 节里编写的登录界面吗？这里也是直接拿来用就好了，这下可省了我们不少的功夫。新建布局文件 login.xml，代码如下所示：

```

<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1" >

```

```

<TableRow>

    <TextView
        android:layout_height="wrap_content"
        android:text="Account:" />

    <EditText
        android:id="@+id/account"
        android:layout_height="wrap_content"
        android:hint="Input your account" />
</TableRow>

<TableRow>

    <TextView
        android:layout_height="wrap_content"
        android:text="Password:" />

    <EditText
        android:id="@+id/password"
        android:layout_height="wrap_content"
        android:inputType="textPassword" />
</TableRow>

<TableRow>

    <Button
        android:id="@+id/login"
        android:layout_height="wrap_content"
        android:layout_span="2"
        android:text="Login" />
</TableRow>

</TableLayout>

```

以上代码都是直接复用之前写好的内容，非常开心。不过从这里开始，我们又需要靠自己动手实现了。现在登录界面的布局已经完成，那么接下来就应该去编写登录界面的活动了，新建 LoginActivity 继承自 BaseActivity，代码如下所示：

```
public class LoginActivity extends BaseActivity {

    private EditText accountEdit;

    private EditText passwordEdit;

    private Button login;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);
        accountEdit = (EditText) findViewById(R.id.account);
        passwordEdit = (EditText) findViewById(R.id.password);
        login = (Button) findViewById(R.id.login);
        login.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                String account = accountEdit.getText().toString();
                String password = passwordEdit.getText().toString();
                // 如果账号是admin且密码是123456，就认为登录成功
                if (account.equals("admin") && password.equals("123456")) {
                    Intent intent = new Intent(LoginActivity.this,
MainActivity.class);
                    startActivity(intent);
                    finish();
                } else {
                    Toast.makeText(LoginActivity.this, "account or password
is invalid",
                                Toast.LENGTH_SHORT).show();
                }
            }
        });
    }

}
```

可以看到，这里我们模拟了一个非常简单的登录功能。首先使用 `setContentView()` 方法将 `login` 布局加载进来，并调用 `findViewById()` 方法分别获取到账号输入框、密码输入框以及

登录按钮的实例。然后在登录按钮的点击事件里面对输入的账号和密码进行判断，如果账号是 admin 并且密码是 123456，就认为登录成功并跳转到 MainActivity，否则就提示用户账号或密码错误。

因此，你就可以将 MainActivity 理解成是登录成功后进入的程序主界面了，这里我们并不需要在主界面里提供什么花哨的功能，只需要加入强制下线功能就可以了，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/force_offline"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send force offline broadcast" />

</LinearLayout>
```

非常简单，只有一个按钮而已。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends BaseActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button forceOffline = (Button) findViewById(R.id.force_offline);
        forceOffline.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent("com.example.broadcastbestpractice.
FORCE_OFFLINE ");
                sendBroadcast(intent);
            }
        });
    }

}
```

同样非常简单，不过这里有个重点，我们在按钮的点击事件里面发送了一条广播，广播

的值为 `com.example.broadcastbestpractice.FORCE_OFFLINE`，这条广播就是用于通知程序强制用户下线的。也就是说强制用户下线的逻辑并不是写在 `MainActivity` 里的，而是应该写在接收这条广播的广播接收器里面，这样强制下线的功能就不会依附于任何的界面，不管是在程序的任何地方，只需要发出这样一条广播，就可以完成强制下线的操作了。

那么毫无疑问，接下来我们就需要创建一个广播接收器了，新建 `ForceOfflineReceiver` 继承自 `BroadcastReceiver`，代码如下所示：

```
public class ForceOfflineReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(final Context context, Intent intent) {
        AlertDialog.Builder dialogBuilder = new AlertDialog.Builder(context);
        dialogBuilder.setTitle("Warning");
        dialogBuilder.setMessage("You are forced to be offline. Please try
to login again.");
        dialogBuilder.setCancelable(false);
        dialogBuilder.setPositiveButton("OK",
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    ActivityCollector.finishAll(); // 销毁所有活动
                    Intent intent = new Intent(context,
LoginActivity.class);
                    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                    context.startActivity(intent); // 重新启动LoginActivity
                }
            });
        AlertDialog alertDialog = dialogBuilder.create();
        // 需要设置AlertDialog的类型，保证在广播接收器中可以正常弹出

        alertDialog.getWindow().setType(WindowManager.LayoutParams.TYPE_SYSTEM_ALERT);
        alertDialog.show();
    }

}
```

这次 `onReceive()` 方法里可不再是仅仅弹出一个 `Toast` 了，而是加入了较多的代码，那我们就来仔细地看一下吧。首先肯定是使用 `AlertDialog.Builder` 来构建一个对话框，注意这里一定要调用 `setCancelable()` 方法将对话框设为不可取消，否则用户按一下 `Back` 键就可以关闭

对话框继续使用程序了。然后使用 `setPositiveButton()` 方法来给对话框注册确定按钮，当用户点击了确定按钮时，就调用 `ActivityCollector` 的 `finishAll()` 方法来销毁掉所有活动，并重新启动 `LoginActivity` 这个活动。另外，由于我们是在广播接收器里启动活动的，因此一定要给 `Intent` 加入 `FLAG_ACTIVITY_NEW_TASK` 这个标志。最后，还需要把对话框的类型设为 `TYPE_SYSTEM_ALERT`，不然它将无法在广播接收器里弹出。

这样的话，所有强制下线的逻辑就已经完成了，接下来我们还需要对 `AndroidManifest.xml` 文件进行配置，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcastbestpractice"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="19" />

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".LoginActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MainActivity" >
        </activity>
        <receiver android:name=".ForceOfflineReceiver" >
            <intent-filter>
                <action android:name="com.example.broadcastbestpractice.
FORCE_OFFLINE" />
            </intent-filter>
        </receiver>
```



```
</application>
</manifest>
```

这里有几内容需要注意，首先由于我们在 ForceOfflineReceiver 里弹出了一个系统级别的对话框，因此必须要声明 android.permission.SYSTEM_ALERT_WINDOW 权限。然后对 LoginActivity 进行注册，并把它设置为主活动，因为肯定不能让用户启动程序就直接进入 MainActivity 吧。最后再对 ForceOfflineReceiver 进行注册，并指定它接收 com.example.broadcastbestpractice.FORCE_OFFLINE 这条广播。

好了，现在来尝试运行一下程序吧，首先会进入到登录界面，并可以在这里输入账号和密码，如图 5.10 所示。

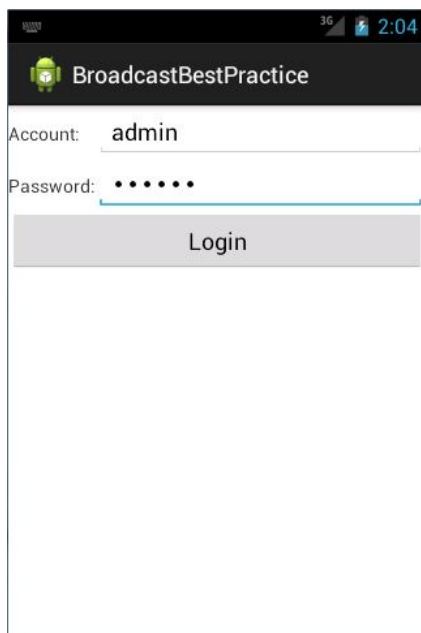


图 5.10

如果输入的账号是 admin，密码是 123456，点击登录按钮就会进入到程序的主界面，如图 5.11 所示。



图 5.11

这时点击一下发送广播的按钮，就会发出一条强制下线的广播，ForceOfflineReceiver 里收到这条广播后会弹出一个对话框提示用户已被强制下线，如图 5.12 所示。

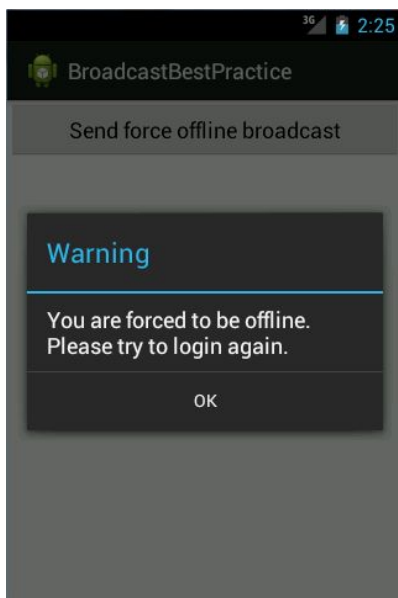


图 5.12

这时用户将无法再对界面的任何元素进行操作，只能点击确定按钮，然后会重新回到登录界面。这样，强制下线功能就已经完整地实现了。

结束了本章的最佳实践部分，接下来我们要进入一个特殊的环节。相信你一定也知道，几乎所有出色的项目都不会是由一个人单枪匹马完成的，而是由一个团队共同合作开发完成的。这个时候多人之间代码同步的问题就显得异常重要，因此版本控制工具也是应运而生。常见的版本控制工具主要有 `svn` 和 `Git`，本书中将会对 `Git` 的使用方法进行全面的讲解，并且讲解的内容是穿插于一些章节当中的。那么今天，我们就先来看一看关于 `Git` 最基本的用法。

经验值：+5000

目前经验值：24905

级别：鸟

赢得宝物：来神界已经有些时日了，回想过去数周发生的事情，我感慨万千，我，一个人界的小白，无意中拾取了一张神界的 `Android` 前辈遗失的修行卡，而误打误撞穿过了人神界面来到神界，修行这意味着无上荣光的 `Android` 开发技术，从最初的萌级小菜鸟，到小菜鸟，到菜鸟，到如今的鸟，我走过了怎样的路，我吃尽了多少苦，服用了多少带有副作用的大力丸，还有为了打败那些庞大的对手（它们的屁股看起来比我家的房子都大），我吞下了多少味道不佳的恢复剂、怒吼丸、鄙视丸……，想想它们都是用什么做的，我就想吐，这些，所有这些，只有我自己知道。想到这里，我热泪盈眶，我被我自己感动了。因此，面对着眼前的这张张牙舞爪的告示，我决定要挑战一下自己，这是神界最大的程序员开发组织 `TNND` 发布的一张 `Android` 开发挑战赛的告示，奖品丰厚、极其嚣张，特等奖是三界意念扭曲场发生器、一等奖是核聚变级大脑编程能力增强器、二等奖是行星级代码重构卡车一辆，三等奖是超高速 `bug` 修复加特林机关枪。纪念品也很丰富，说都是有益身心、可增强编程能力的好东西，但为了保持神秘感，暂不透露。我一开始盯住的是纪念品，上面写着凡报名者均可领取，报名条件是要求达到菜鸟级，这个我很有把握，但我转念一想，难道我就只配领个纪念品么，不，我来就是为了大奖而来的。我对自己说，你行滴！我交了不算便宜的报名费，领到了纪念品，一颗大型信心增大力丸、一个双肩包（印有“`TNND`，你身边的好伙伴”字样）、2瓶恢复剂、3颗怒吼丸，面对着这一坨东西，我情绪有些低落，赶紧服下了大型信心增大力丸，平复了一下心情，踏进了赛场。赛场内人山人海，你绝想不到在神界也有这么多玩 `Android` 的，大部分都是菜鸟、小菜鸟和萌级小菜鸟，因为他们脸上都带着刚服用了大型信心增大力丸后特有的那种相当僵硬的喜悦之情。比赛过程就不细说了，总之，经过两轮搏命厮杀，我成功干掉了一只资深鸟，但随即被一只头领鸟毫无悬念的拿下，我写的程序在他面前根本走不上一个回合。大部分和我一起参赛的鸟级以下选手都迅速被淘汰出局，大家垂头丧气熙熙攘攘地走出了赛场，药效已过，出来的时候人很多，但当我走出镇子时身边已几乎没有人。出发时同行的人再多，最终也都会分道扬镳。前路漫漫，我提了提手中的哨棒（这又谁啊？又给我塞哨棒！靠）。已好久没有出现的那只怪松鼠又再次出现在路旁，它

小爪子握成拳头样，然后猛地往下一顿，这是在对我加油么？这个怪松鼠已经跟了我很久了，我决定上前问问它到底想干嘛，但它打了个响指就消失了。我没有停，继续走到了它刚才出现的位置，在那儿坐了一会儿，吃了点干粮，喝了点水，我感觉体力恢复了些。继续前进。

5.6 Git 时间，初识版本控制工具

Git 是一个开源的分布式版本控制工具，它的开发者就是鼎鼎大名的 Linux 操作系统的作者 Linus Torvalds。Git 被开发出来的初衷本是为了更好地管理 Linux 内核，而现在却早已被广泛应用于全球各种大中小型的项目中。今天是我们关于 Git 的第一堂课，主要是讲解一下它最基本的用法，那么就从安装 Git 开始吧。

5.6.1 安装 Git

由于 Git 和 Linux 操作系统都是同一个作者，因此不用我说你也应该猜到 Git 在 Linux 上的安装是最简单方便的。比如你使用的是 Ubuntu 系统，只需要打开 shell 界面，并输入：

```
sudo apt-get install git-core
```

按下回车后输入密码，即可完成 Git 的安装。

不过我相信你更有可能使用的还是 Windows 操作系统，因此本小节的重点是教会你如何在 Windows 上安装 Git。不同于 Linux，Windows 上可无法通过一行命令就完成安装了，我们需要先把 Git 的安装包下载下来。访问网址 <http://msysgit.github.io/>，可以看到如图 5.13 所示的页面。

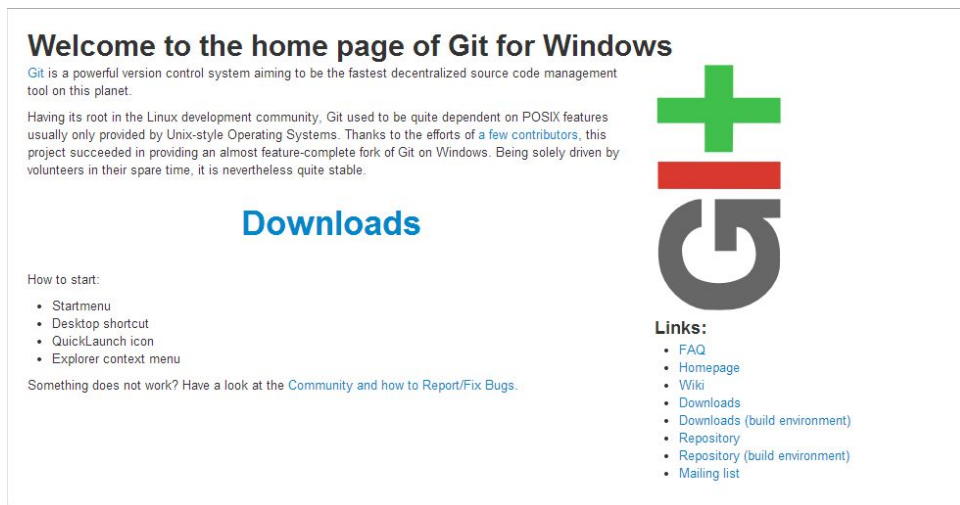


图 5.13

点击网页中央的 Downloads 链接会跳转到 Google Code 的下载列表页面,如图 5.14 所示。


	Git-1.8.4-preview20130916.exe	Full installer for official Git for Windows 1.8.4	Beta
	Git-1.8.3-preview20130601.exe	Full installer for official Git for Windows 1.8.3	Beta
	Git-1.8.1.2-preview20130201.exe	Full installer for official Git for Windows 1.8.1.2	Beta
	Git-1.8.0-preview20121022.exe	Full installer for official Git for Windows 1.8.0	Beta
	Git-1.7.11-preview20120710.exe	Full installer for official Git for Windows 1.7.11	Beta
	Git-1.7.11-preview20120704.exe	Full installer for official Git for Windows 1.7.11	Beta
	Git-1.7.10-preview20120409.exe	Full installer for official Git for Windows 1.7.10	Beta
	Git-1.7.9-preview20120201.exe	Full installer for official Git for Windows 1.7.9	Beta
	Git-1.7.8-preview20111206.exe	Full installer for official Git for Windows 1.7.8	Beta

图 5.14

这里目前最新的 Git 版本是 1.8.4,我就准备使用这一版本了,如果你下载的时候发现又有新的版本,可以尝试一下最新版本的 Git。点击左侧的向下箭头按钮可以开始下载,下载完成后双击安装包进行安装,之后一直点击下一步就可以完成安装了。

5.6.2 创建代码仓库

虽然在 Windows 上安装的 Git 是可以在图形界面上进行操作的,但是这里我并不建议你使用这一功能,因为 Git 的各种命令才是你应该掌握的核心技能,并且不管你是在哪个操作系统中,使用命令来操作 Git 肯定都是通用的。

那么我们现在就来尝试一下如何通过命令来使用 Git,如果你使用的是 Linux 系统,就先打开 shell 界面,如果使用的是 Windows 系统,就从开始里找到 Git Bash 并打开。

首先应该配置一下你的身份,这样在提交代码的时候 Git 就可以知道是谁提交的了,命令如下所示:

```
git config --global user.name "Tony"
git config --global user.email "tony@gmail.com"
```

配置完成后你还可以使用同样的命令来查看是否配置成功,只需要将最后的名字和邮箱地址去掉即可,如图 5.15 所示。

```
Tony@TONY-PC ~
$ git config --global user.name
Tony
Tony@TONY-PC ~
$ git config --global user.email
tony@gmail.com
```

图 5.15

然后我们就可以开始创建代码仓库了，仓库（Repository）是用于保存版本管理所需信息的地方，所有本地提交的代码都会被提交到代码仓库中，如果有需要还可以再推送到远程仓库中。

这里我们尝试着给 BroadcastBestPractice 项目建立一个代码仓库。先进入到 BroadcastBestPractice 项目的目录下面，如图 5.16 所示。

```
Tony@TONY-PC /f
$ cd f:

Tony@TONY-PC /f
$ cd codes/AndroidFirstLine/BroadcastBestPractice/

Tony@TONY-PC /f/codes/AndroidFirstLine/BroadcastBestPractice
$
```

图 5.16

然后在这个目录下面输入如下命令：

git init

很简单吧！只需要一行命令就可以完成创建代码仓库的操作，如图 5.17 所示。

```
Tony@TONY-PC /f/codes/AndroidFirstLine/BroadcastBestPractice
$ git init
Initialized empty Git repository in f:/codes/AndroidFirstLine/BroadcastBestPractice/.git/
```

图 5.17

仓库创建完成后，会在 BroadcastBestPractice 项目的根目录下生成一个隐藏的 .git 文件夹，这个文件夹就是用来记录本地所有的 Git 操作的，可以通过 ls -al 命令来查看一下，如图 5.18 所示。

```
Tony@TONY-PC /f/codes/AndroidFirstLine/BroadcastBestPractice (master)
$ ls -al
total 39
drwxr-xr-x 15 Tony Administ 4096 Sep 15 17:52 .
drwxr-xr-x 17 Tony Administ 4096 Sep 12 20:19 ..
-rw-r--r-- 1 Tony Administ 475 Sep 12 20:20 .classpath
drwxr-xr-x 12 Tony Administ 4096 Sep 15 17:52 .git
-rw-r--r-- 1 Tony Administ 857 Sep 12 20:19 .project
-rw-r--r-- 1 Tony Administ 1223 Sep 14 21:38 AndroidManifest.xml
drwxr-xr-x 2 Tony Administ 0 Sep 12 20:19 assets
drwxr-xr-x 10 Tony Administ 4096 Sep 14 22:02 bin
drwxr-xr-x 3 Tony Administ 0 Sep 12 20:20 gen
drwxr-xr-x 1 Tony Administ 51394 Sep 12 20:19 ic_launcher-web.png
drwxr-xr-x 3 Tony Administ 0 Sep 12 20:19 libs
-rw-r--r-- 1 Tony Administ 781 Sep 12 20:19 proguard-project.txt
-rw-r--r-- 1 Tony Administ 563 Sep 14 21:38 project.properties
drwxr-xr-x 14 Tony Administ 4096 Sep 12 20:19 res
drwxr-xr-x 3 Tony Administ 0 Sep 12 20:19 src
```

图 5.18

如果你想要删除本地仓库，只需要删除这个文件夹就行了。

5.6.3 提交本地代码

代码仓库建立完之后就可以提交代码了，其实提交代码的方法也非常简单，只需要使用 `add` 和 `commit` 命令就可以了。`add` 是用于把想要提交的代码先添加进来，而 `commit` 则是真正地去执行提交操作。比如我们想添加 `AndroidManifest.xml` 文件，就可以输入如下命令：

```
git add AndroidManifest.xml
```

这是添加单个文件的方法，那如果我们想添加某个目录呢？其实只需要在 `add` 后面加上目录名就可以了。比如将整个 `src` 目录下的所有文件都进行添加，就可以输入如下命令：

```
git add src
```

可是这样一个个地添加感觉还是有些复杂，有没有什么办法可以一次性就把所有的文件都添加好呢？当然可以，只需要在 `add` 的后面加上一个点，就表示添加所有的文件了，命令如下所示：

```
git add .
```

现在 `BroadcastBestPractice` 项目下所有的文件都已经添加好了，我们可以来提交一下了，输入如下命令：

```
git commit -m "First commit."
```

注意在 `commit` 命令的后面我们一定要通过 `-m` 参数来加上提交的描述信息，没有描述信息的提交被认为是不合法的。这样所有的代码就已经成功提交了！

好了，关于 `Git` 的内容，今天我们就学到这里，虽然内容并不多，但是你已经将 `Git` 最基本的用法都掌握了，不是吗？在本书后面的章节，还会穿插一些 `Git` 的讲解，到时候你将学会更多关于 `Git` 的使用技巧，现在就让我们来总结一下吧。

5.7 小结与点评

本章中我们主要是对 `Android` 的广播机制进行了深入的研究，不仅了解了广播的理论知识、还掌握了接收广播、发送自定义广播以及本地广播的使用方法。广播接收器是属于 `Android` 四大组件之一，在不知不觉中你已经掌握了四大组件中的两个了。

在最佳实践环节中你一定也收获了不少，不仅运用到了本章所学的广播知识，还将前面章节所学到的技巧综合运用到了一起。经过这个例子之后，相信你对所涉及到的每个知识点都有了更深一层的认识。另外，本章还添加了一个最最特殊的环节，即 `Git` 时间。在这个环节中我们对 `Git` 这个版本控制工具进行了初步的学习，后面还会学习关于它的更多内容。

下一章我们本应该继续学习 Android 中剩余的四大组件，不过由于学习内容提供器之前需要先掌握 Android 中的持久化技术，因此下一章我们就先对这一主题展开讨论。

经验值：+4000 升级！（由鸟升级至资深鸟） 目前经验值：23905

级别：资深鸟

获赠宝物：拜访 Git 领主。获赠智能版本控制器一个、犀牛皮 Android 战袍一套。Git 领主是一位明君，将整个领地治理得井井有条，并且热情好客，极有古风，对走在神界 Android 开发之路上的途经此地的后辈们总会慷慨解囊，给予帮助。就在我升级成资深鸟后，我突然发现我周围亮了一点，原来一旦达到资深鸟级别，身体会开始微微发光！作为一个人界的普通人，一名屌丝，作为众多每日挤地铁、挤公交，造就女神送给土豪的屌丝中的一员，我被自己这一身皮囊竟然能发出光芒这一事实惊得呆立当场，十几秒钟之后才慢慢回过神来，此时夜已深，我突然想起了什么，我掏出了《算法本源》，我自身的光芒照在上面，虽然很微弱，但隐约能分辨字迹，我发现我能够读懂了。以前毫无头绪的表述现在开始产生意义了。虽然只能发出一点点亮光，虽然只在周围很暗的情况下才能看到这光芒，但毕竟这是我自身发出的光芒啊！在最黑暗处我照亮了自己。我相信，随着我级别的提升，我还可以照亮他人。从这一刻开始，我要重新认识自己。满天星斗。我继续前进。