Mobile Applications (420-P84-AB)

John Abbott College

Intents

Aakash Malhotra

Intent

An `Intent` is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use cases:

Starting an activity

- An `Activity` represents a single screen in an app. You can start a new instance of an `Activity` by passing an `Intent` to `startActivity()`. The `Intent` describes the activity to start and carries any necessary data.
- If you want to receive a result from the activity when it finishes, call `startActivityForResult()`. Your activity receives the result as a separate `Intent` object in your activity's `onActivityResult()` callback..

Intent

Starting a service

- A `Service` is a component that performs operations in the background without a user interface. With Android 5.0 (API level 21) and later, you can start a service with `JobScheduler`.
- For versions earlier than Android 5.0 (API level 21), you can start a service by using methods of the `Service` class. You can start a service to perform a one-time operation (such as downloading a file) by passing an `Intent` to `startService()`. The `Intent` describes the service to start and carries any necessary data.
- If the service is designed with a client-server interface, you can bind to the service from another component by passing an `Intent` to `bindService()`.

Intent

Delivering a broadcast

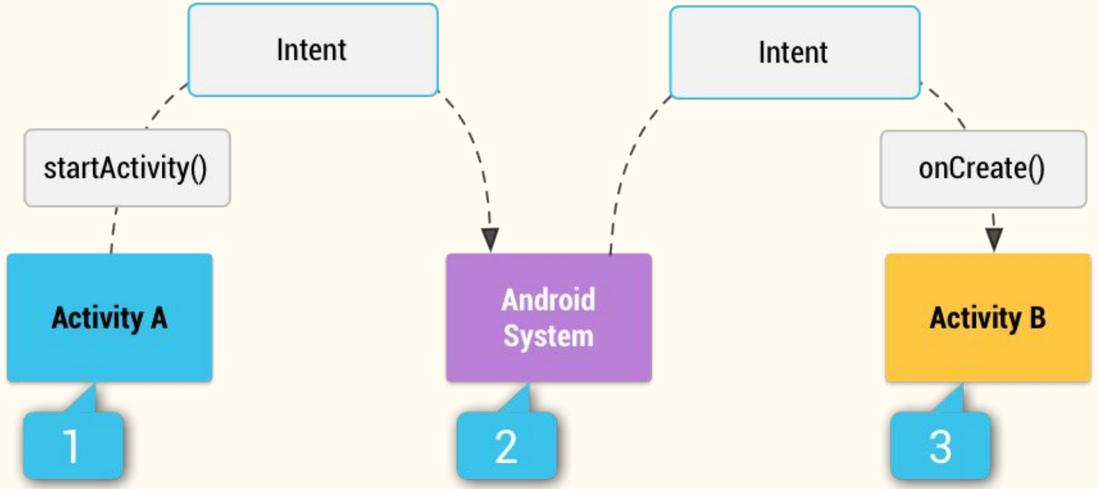
- A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an `Intent` to `sendBroadcast()` or `sendOrderedBroadcast()`.

Intent Types

There are two types of intents:

- **Explicit intents** specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name. You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.
- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Figure shows how an intent is used when starting an activity. When the `Intent` object names a specific activity component explicitly, the system immediately starts that component.



How an implicit intent is delivered through the system to start another activity: **[1]** *Activity A* creates an `Intent` with an action description and passes it to `startActivity()`. **[2]** The Android System searches all apps for an intent filter that matches the intent. When a match is found, **[3]** the system starts the matching activity (*Activity B*) by invoking its `onCreate()` method and passing it the `Intent`.

When you use an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the *intent filters* declared in the manifest file of other apps on the device. If the intent matches an intent filter, the system starts that component and delivers it the `Intent` object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

Intent Filters:

An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive. For instance, by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent. Likewise, if you do *not* declare any intent filters for an activity, then it can be started only with an explicit intent.

Caution: To ensure that your app is secure, always use an explicit intent when starting a **`Service`** and do not declare intent filters for your services. Using an implicit intent to start a service is a security hazard because you can't be certain what service will respond to the intent, and the user can't see which service starts. Beginning with Android 5.0 (API level 21), the system throws an exception if you call **`bindService()`** with an implicit intent.

Building an Intent

An `Intent` object carries information that the Android system uses to determine which component to start (such as the exact component name or component category that should receive the intent), plus information that the recipient component uses in order to properly perform the action (such as the action to take and the data to act upon).

The primary information contained in an `Intent` is the following:

Component name

> The name of the component to start.
>
> This is optional, but it's the critical piece of information that makes an intent *explicit*, meaning that the intent should be delivered only to the app component defined by the component name. Without a component name, the intent is *implicit* and the system decides which component should receive the intent based on the other intent information (such as the action, data, and category—described below).

Building an Intent

If you need to start a specific component in your app, you should specify the component name.

This field of the `Intent` is a `ComponentName` object, which you can specify using a fully qualified class name of the target component, including the package name of the app, for example, `com.example.ExampleActivity`. You can set the component name with `setComponent()`, `setClass()`, `setClassName()`, or with the `Intent` constructor.

Action

> A string that specifies the generic action to perform (such as *view* or *pick*).
>
> In the case of a broadcast intent, this is the action that took place and is being reported. The action largely determines how the rest of the intent is structured—particularly the information that is contained in the data and extras.

Building an Intent

You can specify your own actions for use by intents within your app (or for use by other apps to invoke components in your app), but you usually specify action constants defined by the `Intent` class or other framework classes. Here are some common actions for starting an activity:

`ACTION_VIEW`

Use this action in an intent with `startActivity()` when you have some information that an activity can show to the user, such as a photo to view in a gallery app, or an address to view in a map app.

`ACTION_SEND`

Also known as the *share* intent, you should use this in an intent with `startActivity()`when you have some data that the user can share through another app, such as an email app or social sharing app.

Building an Intent

See the `Intent` class reference for more constants that define generic actions. Other actions are defined elsewhere in the Android framework, such as in `Settings` for actions that open specific screens in the system's Settings app.

You can specify the action for an intent with `setAction()` or with an `Intent` constructor.

Data

The URI (a `Uri` object) that references the data to be acted on and/or the MIME type of that data. The type of data supplied is generally dictated by the intent's action. For example, if the action is `ACTION_EDIT`, the data should contain the URI of the document to edit.

When creating an intent, it's often important to specify the type of data (its MIME type) in addition to its URI. For example, an activity that's able to display images probably won't be able to play an audio file, even though the URI formats could be similar. Specifying the MIME type of your data helps the Android system find the best component to receive your intent.

Building an Intent

However, the MIME type can sometimes be inferred from the URI—particularly when the data is a `content:` URI. A `content:` URI indicates the data is located on the device and controlled by a `ContentProvider`, which makes the data MIME type visible to the system.

To set only the data URI, call `setData()`. To set only the MIME type, call `setType()`. If necessary, you can set both explicitly with `setDataAndType()`.

Caution: If you want to set both the URI and MIME type, *don't* call **`setData()`** and **`setType()`** because they each nullify the value of the other. Always use **`setDataAndType()`** to set both URI and MIME type.

Building an Intent

Category

A string containing additional information about the kind of component that should handle the intent. Any number of category descriptions can be placed in an intent, but most intents do not require a category. Here are some common categories:

`CATEGORY_BROWSABLE`

The target activity allows itself to be started by a web browser to display data referenced by a link, such as an image or an e-mail message.

`CATEGORY_LAUNCHER`

The activity is the initial activity of a task and is listed in the system's application launcher.

You can specify a category with `addCategory()`.

Building an Intent

These properties listed above (component name, action, data, and category) represent the defining characteristics of an intent. By reading these properties, the Android system is able to resolve which app component it should start. However, an intent can carry additional information that does not affect how it is resolved to an app component. An intent can also supply the following information:

Extras

Key-value pairs that carry additional information required to accomplish the requested action. Just as some actions use particular kinds of data URIs, some actions also use particular extras.

Flags

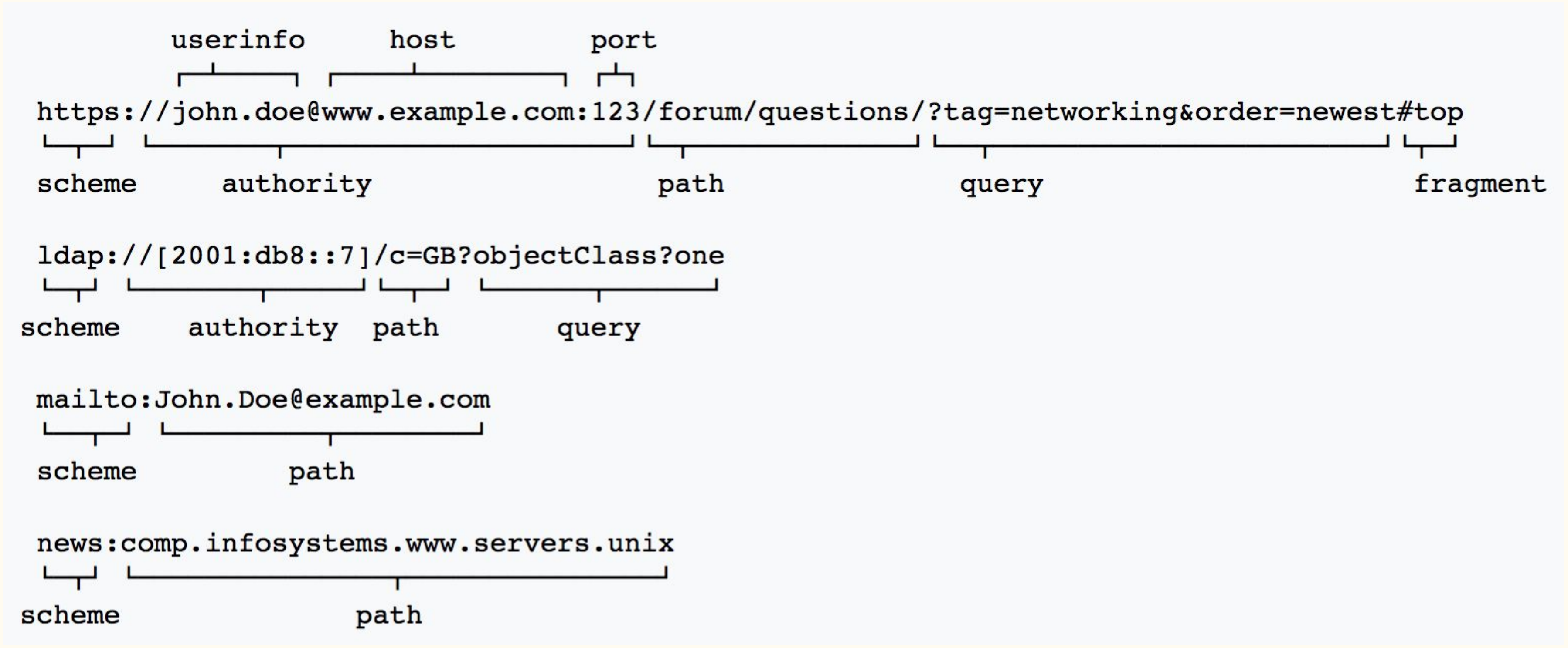
Flags are defined in the `Intent` class that function as metadata for the intent. The flags may instruct the Android system how to launch an activity (for example, which task the activity should belong to) and how to treat it after it's launched (for example, whether it belongs in the list of recent activities).

What is a Uri

A **Uniform Resource Identifier** (**URI**) is a string of characters designed for unambiguous identification of resources and extensibility via the URI scheme.

Uri.Builder can be used to build Uri, less error prone way of creating a Uri.

What is a Uri



userinfo
host
port
https://john.doe@www.example.com:123/forum/questions/?tag=networking&order=newest#top
scheme
authority
path
query
fragment
ldap://[2001:db8::7]/c=GB?objectClass?one
scheme
authority
path
query
mailto:John.Doe@example.com
scheme
path
news:comp.infosystems.www.servers.unix
scheme
path

Tasks:

Use one menu option for each feature

Create an application for John Abbott that has a menu items to:

- open its location on the map (hint: use the official documentation)
- open the official website
- display page that gives information about the college

References:

- https://developer.android.com/guide/components/intents-filters
- https://developer.android.com/guide/components/intents-common
- https://developer.android.com/reference/android/content/Intent