



通用 Intent

本文内容

[显示详细信息](#)

- [闹钟](#)
- [日历](#)
- [相机](#)
- [联系人/人员应用](#)
- [电子邮件](#)
- [文件存储](#)
- [本地操作](#)
- [地图](#)
- [音乐或视频](#)
- [新笔记](#)
- [电话](#)
- [搜索](#)
- [设置](#)
- [发送短信](#)
- [网络浏览器](#)
- [使用 Android 调试桥验证 Intent](#)

另请参阅

- [Intent 和 Intent 过滤器](#)

Intent 用于通过描述您想在某个 [Intent](#) 对象中执行的简单操作（如“查看地图”或“拍摄照片”）来启动另一应用中的某个 Activity。这种 Intent 称作隐式 Intent，因为它并不指定要启动的应用组件，而是指定一项操作并提供执行该操作所需的一些数据。

当您调用 `startActivity()` 或 `startActivityForResult()` 并向其传递隐式 Intent 时，系统会将 [Intent 解析](#) 为可处理该 Intent 的应用并启动其对应的 [Activity](#)。如果有多个应用可处理 Intent，系统会为用户显示一个对话框，供其选择要使用的应用。

本页面介绍几种可用于执行常见操作的隐式 Intent，按处理 Intent 的应用类型分成不同部分。此外，每个部分还介绍如何创建 [Intent 过滤器](#) 来公布您的应用执行相应操作的能力。

注意：如果设备上没有可接收隐式 Intent 的应用，您的应用将在调用 `startActivity()` 时崩溃。如需事先验证是否存在可接收 Intent 的应用，请对 [Intent](#) 对象调用 `resolveActivity()`。如果结果为非空，则至少有一个应用能够处理该 Intent，并且可以安全调用 `startActivity()`。如果结果为空，则您不应使用该 Intent。如有可能，您应停用调用该 Intent 的功能。

如果您不熟悉如何创建 Intent 或 Intent 过滤器，应该先阅读 [Intent 和 Intent 过滤器](#)。

如需了解如何从开发主机触发本页面上所列的 Intent，请参阅[使用 Android 调试桥验证 Intent](#)。

Google Voice Actions


[Google Voice Actions](#) 会触发本页面上所列的一些 Intent 来响应语音命令。如需了解详细信息，请参阅[Google Voice Actions 触发的 Intent](#)。

闹钟

创建闹铃

如需创建新闹铃，请使用 `ACTION_SET_ALARM` 操作并使用下文介绍的 extra 指定时间和消息等闹铃详细信息。

注：Android 2.3（API 级别 9）及更高版本上只提供了小时、分钟和消息 extra。其他 extra 是在更新版本的平台上新增的 extra。

 Google Voice Actions

- “设置一个上午 7 点的闹铃”

操作

`ACTION_SET_ALARM`

数据 URI

无

MIME 类型

无

Extra

`EXTRA_HOUR`

闹铃的小时。

`EXTRA_MINUTES`

闹铃的分钟。

`EXTRA_MESSAGE`

用于标识闹铃的自定义消息。

`EXTRA_DAYS`

一个 `ArrayList`，其中包括应重复触发该闹铃的每个周日。 每一天都必须使用 `Calendar` 类中的某个整型值（如 `MONDAY`）进行声明。

对于一次性闹铃，无需指定此 extra。

`EXTRA_RINGTONE`

一个 `content: URI`，用于指定闹铃使用的铃声，也可指定 `VALUE_RINGTONE_SILENT` 以不使用铃声。

如需使用默认铃声，则无需指定此 extra。

`EXTRA_VIBRATE`

一个布尔型值，用于指定该闹铃触发时是否振动。

`EXTRA_SKIP_UI`

一个布尔型值，用于指定响应闹铃的应用在设置闹铃时是否应跳过其 UI。 若为 `true`，则应用应跳过任何确认 UI，直接设置指定的闹铃。

示例 Intent：

```
public void createAlarm(String message, int hour, int minutes) {
    Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM)
        .putExtra(AlarmClock.EXTRA_MESSAGE, message)
        .putExtra(AlarmClock.EXTRA_HOUR, hour)
        .putExtra(AlarmClock.EXTRA_MINUTES, minutes);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

注：

为了调用 `ACTION_SET_ALARM` Intent，您的应用必须具有 `SET_ALARM` 权限：

```
<uses-permission android:name="com.android.alarm.permission.SET_ALARM" />
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SET_ALARM" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

创建定时器

如需创建倒计时器，请使用 `ACTION_SET_TIMER` 操作并使用下文介绍的 `extra` 指定持续时间等定时器详细信息。

注：此 Intent 是在 Android 4.4（API 级别 19）中添加的。



Google Voice Actions

- “设置 5 分钟定时器”

操作

`ACTION_SET_TIMER`

数据 URI

无

MIME 类型

无

Extra

`EXTRA_LENGTH`

以秒为单位的定时器定时长度。

`EXTRA_MESSAGE`

用于标识定时器的自定义消息。

`EXTRA_SKIP_UI`

一个布尔型值，用于指定响应定时器的应用在设置定时器时是否应跳过其 UI。若为 `true`，则应用应跳过任何确认 UI，直接启动指定的定时器。

示例 Intent：

```
public void startTimer(String message, int seconds) {
    Intent intent = new Intent(AlarmClock.ACTION_SET_TIMER)
        .putExtra(AlarmClock.EXTRA_MESSAGE, message)
        .putExtra(AlarmClock.EXTRA_LENGTH, seconds)
        .putExtra(AlarmClock.EXTRA_SKIP_UI, true);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

注：

为了调用 `ACTION_SET_TIMER` Intent，您的应用必须具有 `SET_ALARM` 权限：

```
<uses-permission android:name="com.android.alarm.permission.SET_ALARM" />
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SET_TIMER" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

显示所有闹铃

如需显示闹铃列表，请使用 `ACTION_SHOW_ALARMS` 操作。

尽管调用此 Intent 的应用并不多（使用它的主要是系统应用），但任何充当闹钟的应用都应实现此 Intent 过滤器，并通过显示现有闹铃列表作出响应。

注：此 Intent 是在 Android 4.4（API 级别 19）中添加的。

操作

`ACTION_SHOW_ALARMS`

数据 URI

无

MIME 类型

无

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SHOW_ALARMS" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

日历

添加日历事件

如需向用户的日历添加新事件，请使用 `ACTION_INSERT` 操作指定具有 `Events.CONTENT_URI` 的数据 URI。然后您就可以使用下文介绍的 `extra` 指定事件的各类详细信息。

操作

`ACTION_INSERT`

数据 URI

`Events.CONTENT_URI`

MIME 类型

`"vnd.android.cursor.dir/event"`

Extra

`EXTRA_EVENT_ALL_DAY`

一个布尔型值，指定此事件是否为全天事件。

`EXTRA_EVENT_BEGIN_TIME`

事件的开始时间（从新纪年开始计算的毫秒数）。

`EXTRA_EVENT_END_TIME`

事件的结束时间（从新纪年开始计算的毫秒数）。

`TITLE`

事件标题。

`DESCRIPTION`

事件说明。

`EVENT_LOCATION`

事件地点。

`EXTRA_EMAIL`

以逗号分隔的受邀者电子邮件地址列表。

可使用 `CalendarContract.EventsColumns` 类中定义的常量指定许多其他事件详细信息。

示例 Intent：

```
public void addEvent(String title, String location, Calendar begin, Calendar end) {
    Intent intent = new Intent(Intent.ACTION_INSERT)
        .setData(Events.CONTENT_URI)
        .putExtra(Events.TITLE, title)
        .putExtra(Events.EVENT_LOCATION, location)
        .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, begin)
        .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, end);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
  <intent-filter>
    <action android:name="android.intent.action.INSERT" />
    <data android:mimeType="vnd.android.cursor.dir/event" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

相机

拍摄照片或视频并将其返回

如需打开相机应用并接收拍摄的照片或视频，请使用 [ACTION_IMAGE_CAPTURE](#) 或 [ACTION_VIDEO_CAPTURE](#) 操作。此外，还可在 [EXTRA_OUTPUT](#) extra 中指定您希望相机将照片或视频保存到的 URI 位置。

操作

[ACTION_IMAGE_CAPTURE](#) 或
[ACTION_VIDEO_CAPTURE](#)

数据 URI 架构

无

MIME 类型

无

Extra

[EXTRA_OUTPUT](#)

相机应用应将照片或视频文件保存到的 URI 位置（[Uri](#) 对象形式）。

当相机应用成功将焦点归还给您的 Activity（您的应用收到 [onActivityResult\(\)](#) 回调）时，您可以按通过 [EXTRA_OUTPUT](#) 值指定的 URI 访问照片或视频。

注：当您使用 [ACTION_IMAGE_CAPTURE](#) 拍摄照片时，相机可能还会在结果 [Intent](#) 中返回缩小尺寸的照片副本（缩略图），这个副本以 [Bitmap](#) 形式保存在名为 `"data"` 的 extra 字段中。

示例 Intent：

```

static final int REQUEST_IMAGE_CAPTURE = 1;
static final Uri mLocationForPhotos;

public void capturePhoto(String targetFilename) {
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    intent.putExtra(MediaStore.EXTRA_OUTPUT,
        Uri.withAppendedPath(mLocationForPhotos, targetFilename));
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent, REQUEST_IMAGE_CAPTURE);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        Bitmap thumbnail = data.getParcelable("data");
        // Do other work with full size photo saved in mLocationForPhotos
        ...
    }
}

```

如需了解有关如何使用此 Intent 拍摄照片的详细信息，包括如何创建与输出位置相适应的 [Uri](#)，请阅读[只拍摄照片](#)或[只拍摄视频](#)。

示例 Intent 过滤器：

```

<activity ...>
    <intent-filter>
        <action android:name="android.media.action.IMAGE_CAPTURE" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>

```

处理此 Intent 时，您的 Activity 应检查传入 [Intent](#) 中是否有 [EXTRA_OUTPUT](#) extra，然后将拍摄的图像或视频保存在该 extra 指定的位置，并调用带 [Intent](#) 的 [setResult\(\)](#)，该 Intent 将经过压缩的缩略图包括在名为 "data" 的 extra 中。

以静态图像模式启动相机应用

如需以静态图像模式打开相机应用，请使用 [INTENT_ACTION_STILL_IMAGE_CAMERA](#) 操作。

操作

[INTENT_ACTION_STILL_IMAGE_CAMERA](#)

数据 URI 架构

无

MIME 类型

无

Extra

无

示例 Intent：

```

public void capturePhoto() {
    Intent intent = new Intent(MediaStore.INTENT_ACTION_STILL_IMAGE_CAMERA);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent);
    }
}

```



Google Voice Actions

- “拍摄照片”

示例 Intent 过滤器：

```
<activity ...>
  <intent-filter>
    <action android:name="android.media.action.STILL_IMAGE_CAMERA" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

以视频模式启动相机应用

如需以视频模式打开相机应用，请使用 [INTENT_ACTION_VIDEO_CAMERA](#) 操作。

操作

[INTENT_ACTION_VIDEO_CAMERA](#)

数据 URI 架构

无

MIME 类型

无

Extra

无

示例 Intent：

```
public void capturePhoto() {
    Intent intent = new Intent(MediaStore.INTENT_ACTION_VIDEO_CAMERA);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
  <intent-filter>
    <action android:name="android.media.action.VIDEO_CAMERA" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

联系人/人员应用

选择联系人

如需让用户选择联系人和为您的应用提供对所有联系人信息的访问权限，请使用 [ACTION_PICK](#) 操作，并将 MIME 类型指定为 [Contacts.CONTENT_TYPE](#)。

传送至您的 [onActivityResult\(\)](#) 回调的结果 [Intent](#) 包含指向所选联系人的 [content](#): URI。响应会利用 [Contacts Provider](#) API 为您的应用授予该联系人的临时读取权限，即使您的应用不具备 [READ_CONTACTS](#) 权限也没有关系。

提示：如果您只需要访问某一条联系人信息（如电话号码或电子邮件地址），请改为参见下一节的内容，其中介绍了如何 [选择特定联系人数据](#)。



Google Voice Actions

- “录制视频”

操作

`ACTION_PICK`

数据 URI 架构

无

MIME 类型

`Contacts.CONTENT_TYPE`

示例 Intent：

```
static final int REQUEST_SELECT_CONTACT = 1;

public void selectContact() {
    Intent intent = new Intent(Intent.ACTION_PICK);
    intent.setType(ContactsContract.Contacts.CONTENT_TYPE);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent, REQUEST_SELECT_CONTACT);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_SELECT_CONTACT && resultCode == RESULT_OK) {
        Uri contactUri = data.getData();
        // Do something with the selected contact at contactUri
        ...
    }
}
```

如需了解有关在获得联系人 URI 后如何检索联系人详情的信息，请阅读[检索联系人详情](#)。请谨记，使用以上 Intent 检索联系人 URI 时，读取该联系人的详情**并不需要** `READ_CONTACTS` 权限。

选择特定联系人数据

如需让用户选择某一条联系人信息，如电话号码、电子邮件地址或其他数据类型，请使用 `ACTION_PICK` 操作，并将 MIME 类型指定为下列其中一个内容类型（如 `CommonDataKinds.Phone.CONTENT_TYPE`），以获取联系人的电话号码。

如果您只需要检索一种类型的联系人数据，则将此方法与来自 `ContactsContract.CommonDataKinds` 类的 `CONTENT_TYPE` 配合使用要比使用 `Contacts.CONTENT_TYPE`（如上一部分中所示）更高效，因为结果可让您直接访问所需数据，无需对[联系人提供程序](#)执行更复杂的查询。

传送至您的 `onActivityResult()` 回调的结果 `Intent` 包含指向所选联系人数据的 `content`：URI。响应会为您的应用授予该联系人数据的临时读取权限，即使您的应用不具备 `READ_CONTACTS` 权限也没有关系。

操作

`ACTION_PICK`

数据 URI 架构

无

MIME 类型

`CommonDataKinds.Phone.CONTENT_TYPE`

从有电话号码的联系人中选取。

`CommonDataKinds.Email.CONTENT_TYPE`

从有电子邮件地址的联系人中选取。

`CommonDataKinds.StructuredPostal.CONTENT_TYPE`

从有邮政地址的联系人中选取。

或者 `ContactsContract` 下众多其他 `CONTENT_TYPE` 值中的一个。

示例 Intent：

```
static final int REQUEST_SELECT_PHONE_NUMBER = 1;

public void selectContact() {
    // Start an activity for the user to pick a phone number from contacts
    Intent intent = new Intent(Intent.ACTION_PICK);
    intent.setType(CommonDataKinds.Phone.CONTENT_TYPE);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent, REQUEST_SELECT_PHONE_NUMBER);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_SELECT_PHONE_NUMBER && resultCode == RESULT_OK) {
        // Get the URI and query the content provider for the phone number
        Uri contactUri = data.getData();
        String[] projection = new String[]{CommonDataKinds.Phone.NUMBER};
        Cursor cursor = getContentResolver().query(contactUri, projection,
            null, null, null);
        // If the cursor returned is valid, get the phone number
        if (cursor != null && cursor.moveToFirst()) {
            int numberIndex = cursor.getColumnIndex(CommonDataKinds.Phone.NUMBER);
            String number = cursor.getString(numberIndex);
            // Do something with the phone number
            ...
        }
    }
}
```

查看联系人

如需显示已知联系人的详情，请使用 `ACTION_VIEW` 操作，并使用 `content:` URI 作为 Intent 数据指定联系人。

初次检索联系人 URI 的方法主要有两种：

- 使用 `ACTION_PICK` 返回的联系人 URI，如上一节所示（此方法不需要任何应用权限）
- 直接访问所有联系人的列表，如[检索联系人列表](#)所述（此方法需要 `READ_CONTACTS` 权限）

操作

`ACTION_VIEW`

数据 URI 架构

`content:<URI>`

MIME 类型

无。该类型是从联系人 URI 推断得出。

示例 Intent：

```
public void viewContact(Uri contactUri) {
    Intent intent = new Intent(Intent.ACTION_VIEW, contactUri);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

编辑现有联系人

如需编辑已知联系人，请使用 [ACTION_EDIT](#) 操作，使用 `content:` URI 作为 Intent 数据指定联系人，并将 `extra` 中由常量指定的任何已知联系人信息包括在 `ContactsContract.Intents.Insert` 中。

初次检索联系人 URI 的方法主要有两种：

- 使用 [ACTION_PICK](#) 返回的联系人 URI，如上一节所示（此方法不需要任何应用权限）
- 直接访问所有联系人的列表，如[检索联系人列表](#)所述（此方法需要 [READ_CONTACTS](#) 权限）

操作

[ACTION_EDIT](#)

数据 URI 架构

`content:<URI>`

MIME 类型

该类型是从联系人 URI 推断得出。

Extra

[ContactsContract.Intents.Insert](#) 中定义的一个或多个 `extra`，以便您填充联系人详情字段。

示例 Intent：

```
public void editContact(Uri contactUri, String email) {
    Intent intent = new Intent(Intent.ACTION_EDIT);
    intent.setData(contactUri);
    intent.putExtra(ContactsContract.Intents.Insert.EMAIL, email);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

如需了解有关如何编辑联系人的详细信息，请阅读[使用 Intent 修改联系人](#)。

插入联系人

如需插入新联系人，请使用 [ACTION_INSERT](#) 操作，将 `Contacts.CONTENT_TYPE` 指定为 MIME 类型，并将 `extra` 中由常量指定的任何已知联系人信息包括在 `ContactsContract.Intents.Insert` 中。

操作

[ACTION_INSERT](#)

数据 URI 架构

无

MIME 类型

`Contacts.CONTENT_TYPE`

Extra

`ContactsContract.Intents.Insert` 中定义的一个或多个 extra。

示例 Intent：

```
public void insertContact(String name, String email) {
    Intent intent = new Intent(Intent.ACTION_INSERT);
    intent.setType(Contacts.CONTENT_TYPE);
    intent.putExtra(ContactsContract.Intents.Insert.NAME, name);
    intent.putExtra(ContactsContract.Intents.Insert.EMAIL, email);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

如需了解有关如何插入联系人的详细信息，请阅读[使用 Intent 修改联系人](#)。

电子邮件

撰写带有可选附件的电子邮件

如需撰写电子邮件，请根据其是否包括附件使用以下其中一项操作，并使用下列 extra 键加入收件人和主题等电子邮件详情。

操作

`ACTION_SENDTO`（适用于不带附件）

`ACTION_SEND`（适用于带一个附件）

`ACTION_SEND_MULTIPLE`（适用于带多个附件）

数据 URI 架构

无

MIME 类型

`"text/plain"`

`"*/*"`

Extra

`Intent.EXTRA_EMAIL`

包含所有“主送”收件人电子邮件地址的字符串数组。

`Intent.EXTRA_CC`

包含所有“抄送”收件人电子邮件地址的字符串数组。

`Intent.EXTRA_BCC`

包含所有“密件抄送”收件人电子邮件地址的字符串数组。

`Intent.EXTRA_SUBJECT`

包含电子邮件主题的字符串。

`Intent.EXTRA_TEXT`

包含电子邮件正文的字符串。

`Intent.EXTRA_STREAM`

指向附件的 `Uri`。如果使用的是 `ACTION_SEND_MULTIPLE` 操作，应将其改为包含多个 `Uri` 对象的 `ArrayList`。

示例 Intent：

```
public void composeEmail(String[] addresses, String subject, Uri attachment) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("*/*");
    intent.putExtra(Intent.EXTRA_EMAIL, addresses);
    intent.putExtra(Intent.EXTRA_SUBJECT, subject);
    intent.putExtra(Intent.EXTRA_STREAM, attachment);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

如果您想确保 Intent 只由电子邮件应用（而非其他短信或社交应用）进行处理，则需使用 `ACTION_SENDTO` 操作并加入 `"mailto:"` 数据架构。例如：

```
public void composeEmail(String[] addresses, String subject) {
    Intent intent = new Intent(Intent.ACTION_SENDTO);
    intent.setData(Uri.parse("mailto:")); // only email apps should handle this
    intent.putExtra(Intent.EXTRA_EMAIL, addresses);
    intent.putExtra(Intent.EXTRA_SUBJECT, subject);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <data android:type="*" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SENDTO" />
        <data android:scheme="mailto" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

文件存储

检索特定类型的文件

如需请求用户选择文档或照片等文件并向您的应用返回文件引用，请使用 `ACTION_GET_CONTENT` 操作并指定所需 MIME 类型。向您的应用返回的文件引用对 Activity 的当前生命周期而言是瞬态引用，因此如果您想稍后进行访问，就必须导入可在稍后读取的副本。用户还可利用此 Intent 在进程中创建新文件（例如，用户可以不选择现有照片，而是用相机拍摄新照片）。

传送到您的 `onActivityResult()` 方法的结果 Intent 包括的数据具有指向该文件的 URI。该 URI 可以是任何类型，如 `http:` URI、`file:` URI 或 `content:` URI。不过，如果您想将可选择的文件限定为可从内容提供程序 (`content:` URI) 访问的文件，以及通过 `openFileDescriptor()` 以文件流形式提供的文件，则您应该为 Intent 添加 `CATEGORY_OPENABLE` 类别。

在 Android 4.3 (API 级别 18) 及更高版本上，您还可以通过为 Intent 添加 `EXTRA_ALLOW_MULTIPLE` 并将其设置为 `true`，允许用户选择多个文件。然后您就可以在 `getClipData()` 返回的 `ClipData` 对象中访问每一个选定的文件。

操作

`ACTION_GET_CONTENT`

数据 URI 架构

无

MIME 类型

与用户应选择的文件类型对应的 MIME 类型。

Extra

`EXTRA_ALLOW_MULTIPLE`

一个布尔型值，声明用户是否可以一次选择多个文件。

`EXTRA_LOCAL_ONLY`

一个布尔型值，声明是否返回的文件必须直接存在于设备上，而不是需要从远程服务下载。

类别（可选）

`CATEGORY_OPENABLE`

只返回可通过 `openFileDescriptor()` 以文件流形式表示的“可打开”文件。

用于获取照片的示例 Intent：

```
static final int REQUEST_IMAGE_GET = 1;

public void selectImage() {
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.setType("image/*");
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(intent, REQUEST_IMAGE_GET);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_GET && resultCode == RESULT_OK) {
        Bitmap thumbnail = data.getParcelable("data");
        Uri fullPhotoUri = data.getData();
        // Do work with photo saved at fullPhotoUri
        ...
    }
}
```

用于返回照片的示例 Intent 过滤器：

```
<activity ...>
  <intent-filter>
    <action android:name="android.intent.action.GET_CONTENT" />
    <data android:type="image/*" />
    <category android:name="android.intent.category.DEFAULT" />
    <!-- The OPENABLE category declares that the returned file is accessible
         from a content provider that supports OpenableColumns
         and ContentResolver.openFileDescriptor() -->
    <category android:name="android.intent.category.OPENABLE" />
  </intent-filter>
</activity>
```

打开特定类型的文件

在 Android 4.4 或更高版本上运行时，您可以不必检索必须导入应用的文件副本（使用 `ACTION_GET_CONTENT` 操作），而是使用 `ACTION_OPEN_DOCUMENT` 操作并指定 MIME 类型，请求 *打开* 由另一个应用管理的文件。如果还需要允许用户创建应用可写入的新文档，可改用 `ACTION_CREATE_DOCUMENT` 操作。例如，`ACTION_CREATE_DOCUMENT` Intent 允许用户选择他们想在哪里创建新 PDF 文档（在另一个管理文档存储的应用内），而不是从现有文档中进行选择 — 您的应用随后会收到其可以写入新文档的 URI 位置。

尽管从 `ACTION_GET_CONTENT` 操作传递至您的 `onActivityResult()` 方法的 Intent 可能返回任何类型的 URI，来自 `ACTION_OPEN_DOCUMENT` 和 `ACTION_CREATE_DOCUMENT` 的结果 Intent 始终将所选文件指定为 `DocumentsProvider` 支持的 `content:` URI。您可以通过 `openFileDescriptor()` 打开该文件，并使用 `DocumentsContract.Document` 中的列查询其详细信息。

返回的 URI 会为您的应用授予对文件的长期读取权限（还可能会授予写入权限）。因此，如果您想读取现有文件而不将其副本导入您的应用，或者您想就地打开和编辑文件，特别适合使用 `ACTION_OPEN_DOCUMENT` 操作（而不是使用 `ACTION_GET_CONTENT`）。

您还可以通过为 Intent 添加 `EXTRA_ALLOW_MULTIPLE` 并将其设置为 `true`，允许用户选择多个文件。如果用户只选择一项，您就可以从 `getData()` 检索该项目。如果用户选择多项，则 `getData()` 返回 `null`，此时您必须改为从 `getClipData()` 返回的 `ClipData` 对象检索每个项目。

注：您的 Intent **必须** 指定 MIME 类型，并且**必须** 声明 `CATEGORY_OPENABLE` 类别。必要时，您可以使用 `EXTRA_MIME_TYPES` extra 添加一个 MIME 类型数组来指定多个 MIME 类型 — 如果您这样做，必须将 `setType()` 中的主 MIME 类型设置为 `"*/*"`。

操作

`ACTION_OPEN_DOCUMENT` 或
`ACTION_CREATE_DOCUMENT`

数据 URI 架构

无

MIME 类型

与用户应选择的文件类型对应的 MIME 类型。

Extra

`EXTRA_MIME_TYPES`

与您的应用请求的文件类型对应的 MIME 类型数组。当您使用此 extra 时，必须在 `setType()` 中将主 MIME 类型设置为 `"*/*"`。

`EXTRA_ALLOW_MULTIPLE`

一个布尔型值，声明用户是否可以一次选择多个文件。

`EXTRA_TITLE`

供与 `ACTION_CREATE_DOCUMENT` 配合使用，用于指定初始文件名。

`EXTRA_LOCAL_ONLY`

一个布尔型值，声明是否返回的文件必须直接存在于设备上，而不是需要从远程服务下载。

类别

`CATEGORY_OPENABLE`

只返回可通过 `openFileDescriptor()` 以文件流形式表示的“可打开”文件。

用于获取照片的示例 Intent：

```
static final int REQUEST_IMAGE_OPEN = 1;

public void selectImage() {
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
    intent.setType("image/*");
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    // Only the system receives the ACTION_OPEN_DOCUMENT, so no need to test.
    startActivityForResult(intent, REQUEST_IMAGE_OPEN);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_OPEN && resultCode == RESULT_OK) {
        Uri fullPhotoUri = data.getData();
        // Do work with full size photo saved at fullPhotoUri
        ...
    }
}
```

第三方应用实际上无法通过 `ACTION_OPEN_DOCUMENT` 操作响应 Intent，而是由系统接收此 Intent，然后在统一用户界面中显示各类应用提供的所有文件。

如需在该 UI 中提供您的应用的文件，并允许其他应用打开它们，您必须实现一个 `DocumentsProvider`，并加入一个 `PROVIDER_INTERFACE` Intent 过滤器 (`"android.content.action.DOCUMENTS_PROVIDER"`)。例如：

```
<provider ...
    android:grantUriPermissions="true"
    android:exported="true"
    android:permission="android.permission.MANAGE_DOCUMENTS">
    <intent-filter>
        <action android:name="android.content.action.DOCUMENTS_PROVIDER" />
    </intent-filter>
</provider>
```

如需了解有关如何实现从其他应用打开您的应用管理的文件的详细信息，请阅读 [存储访问框架指南](#)。

本地操作

叫车

如需叫一台出租车，请使用 `ACTION_RESERVE_TAXI_RESERVATION` 操作。

注：应用必须请求用户确认，然后才能完成操作。

操作

`ACTION_RESERVE_TAXI_RESERVATION`

数据 URI

无

MIME 类型

无

Extra

无

示例 Intent：



Google Voice Actions

- “给我叫一台出租车”
- “给我叫一台车”

(仅限 Android Wear)


```
public void callCar() {
    Intent intent = new Intent(ReserveIntents.ACTION_RESERVE_TAXI_RESERVATION);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="com.google.android.gms.actions.RESERVE_TAXI_RESERVATION" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

地图

显示地图上的位置

如需打开地图，请使用 `ACTION_VIEW` 操作，并通过下文介绍的其中一个架构在 Intent 数据中指定位置信息。

操作

`ACTION_VIEW`

数据 URI 架构

`geo:latitude,longitude`

显示给定经度和纬度处的地图。

示例："geo:47.6,-122.3"

`geo:latitude,longitude?z=zoom`

按特定缩放级别显示给定经度和纬度处的地图。缩放级别为 1 时显示以给定 *纬度*、*经度* 为中心的全球地图。最高（最精确）缩放级别为 23。

示例："geo:47.6,-122.3?z=11"

`geo:0,0?q=lat,lng(label)`

显示给定经度和纬度处带字符串标签的地图。

示例："geo:0,0?q=34.99,-106.61(Treasure)"

`geo:0,0?q=my+street+address`

显示“我的街道地址”的位置（可能是具体地址或位置查询）。

示例："geo:0,0?q=1600+Amphitheatre+Parkway%2C+CA"

注：geo URI 中传递的所有字符串都必须编码。例如，字符串 `1st & Pike, Seattle` 应编码为 `1st%20%26%20Pike%2C%20Seattle`。字符串中的空格可使用 `%20` 编码或替换为加号 (+)。

MIME 类型

无

示例 Intent：

```
public void showMap(Uri geoLocation) {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(geoLocation);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <data android:scheme="geo" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

音乐或视频

播放媒体文件

如需播放音乐文件，请使用 `ACTION_VIEW` 操作，并在 Intent 数据中指定文件的 URI 位置。

操作

`ACTION_VIEW`

数据 URI 架构

`file:<URI>`

`content:<URI>`

`http:<URL>`

MIME 类型

`"audio/*"`

`"application/ogg"`

`"application/x-ogg"`

`"application/itunes"`

或者您的应用可能需要的任何其他类型。

示例 Intent：

```
public void playMedia(Uri file) {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(file);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <data android:type="audio/*" />
    <data android:type="application/ogg" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

基于搜索查询播放音乐

如需基于搜索查询播放音乐，请使用 [INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH](#) Intent。应用可能会触发此 Intent 来响应用户的音乐播放语音命令。接收此 Intent 的应用会在其库存音乐内搜索与给定查询匹配的现有内容，并在找到后开始播放该内容。

此 Intent 应该包括 [EXTRA_MEDIA_FOCUS](#) 字符串 extra，以指定预期搜索模式。例如，搜索模式可指定搜索的目标是艺术家姓名还是歌曲名称。



Google Voice Actions

- “播放 michael jackson 的 billie jean”

操作

[INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH](#)

数据 URI 架构

无

MIME 类型

无

Extra

[MediaStore.EXTRA_MEDIA_FOCUS](#)（必需）

表示搜索模式（用户是否在寻找特定艺术家、专辑、歌曲或播放列表）。大多数搜索模式都需要额外的 extra。例如，如果用户有意收听某一首歌曲，Intent 可能需要额外增加三个 extra：歌曲名称、艺术家和专辑。对于 [EXTRA_MEDIA_FOCUS](#) 的每个值，此 Intent 都支持下列搜索模式：

任意 - `"vnd.android.cursor.item/*"`

播放任意音乐。接收 Intent 的应用应该根据智能选择（如用户最后收听的播放列表）播放音乐。

额外 extra：

- [QUERY](#)（必需）- 一个空字符串。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

非结构化 - `"vnd.android.cursor.item/*"`

播放通过非结构化搜索查询找到的特定歌曲、专辑或类型。当应用无法识别用户想要收听的内容类型时，可能会生成一个具有此搜索模式的 Intent。应用应尽可能使用更确切的搜索模式。

额外 extra：

- [QUERY](#)（必需）- 一个包含艺术家、专辑、歌曲名称或类型任意组合的字符串。

类型 - [Audio.Genres.ENTRY_CONTENT_TYPE](#)

播放特定类型的音乐。

额外 extra：

- `"android.intent.extra.genre"` (必需) - 类型。
- `QUERY` (必需) - 类型。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

艺术家 - `Audio.Artists.ENTRY_CONTENT_TYPE`

播放特定艺术家的音乐。

额外 extra：

- `EXTRA_MEDIA_ARTIST` (必需) - 艺术家。
- `"android.intent.extra.genre"` - 类型。
- `QUERY` (必需) - 一个包含艺术家或类型任意组合的字符串。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

专辑 - `Audio.Albums.ENTRY_CONTENT_TYPE`

播放特定专辑的音乐。

额外 extra：

- `EXTRA_MEDIA_ALBUM` (必需) - 专辑。
- `EXTRA_MEDIA_ARTIST` - 艺术家。
- `"android.intent.extra.genre"` - 类型。
- `QUERY` (必需) - 一个包含专辑或艺术家任意组合的字符串。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

歌曲 - `"vnd.android.cursor.item/audio"`

播放特定歌曲。

额外 extra：

- `EXTRA_MEDIA_ALBUM` - 专辑。
- `EXTRA_MEDIA_ARTIST` - 艺术家。
- `"android.intent.extra.genre"` - 类型。
- `EXTRA_MEDIA_TITLE` (必需) - 歌曲名称。
- `QUERY` (必需) - 一个包含专辑、艺术家、类型或名称任意组合的字符串。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

播放列表 - `Audio.Playlists.ENTRY_CONTENT_TYPE`

播放特定播放列表或符合额外 extra 指定的某些条件的播放列表。

额外 extra：

- `EXTRA_MEDIA_ALBUM` - 专辑。
- `EXTRA_MEDIA_ARTIST` - 艺术家。
- `"android.intent.extra.genre"` - 类型。
- `"android.intent.extra.playlist"` - 播放列表。
- `EXTRA_MEDIA_TITLE` - 播放列表所基于的歌曲名称。

- **QUERY**（必需） - 一个包含专辑、艺术家、类型、播放列表或名称任意组合的字符串。始终提供此 extra，以实现向后兼容性：不了解搜索模式的现有应用可将此 Intent 作为非结构化搜索进行处理。

示例 Intent：

如果用户想收听特定艺术家的音乐，搜索应用可生成以下 Intent：

```
public void playSearchArtist(String artist) {
    Intent intent = new Intent(MediaStore.INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH);
    intent.putExtra(MediaStore.EXTRA_MEDIA_FOCUS,
        MediaStore.Audio.Artists.ENTRY_CONTENT_TYPE);
    intent.putExtra(MediaStore.EXTRA_MEDIA_ARTIST, artist);
    intent.putExtra(SearchManager.QUERY, artist);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.media.action.MEDIA_PLAY_FROM_SEARCH" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

处理此 Intent 时，您的 Activity 应通过检查传入 **Intent** 中 **EXTRA_MEDIA_FOCUS** extra 的值来确定搜索模式。您的 Activity 识别出搜索模式后，应该读取该特定搜索模式额外 extra 的值。您的应用随后便可利用这些信息在其库存音乐内进行搜索，以播放与搜索查询匹配的内容。例如：

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    Intent intent = this.getIntent();
    if (intent.getAction().compareTo(MediaStore.INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH) == 0) {

        String mediaFocus = intent.getStringExtra(MediaStore.EXTRA_MEDIA_FOCUS);
        String query = intent.getStringExtra(SearchManager.QUERY);

        // Some of these extras may not be available depending on the search mode
        String album = intent.getStringExtra(MediaStore.EXTRA_MEDIA_ALBUM);
        String artist = intent.getStringExtra(MediaStore.EXTRA_MEDIA_ARTIST);
        String genre = intent.getStringExtra("android.intent.extra.genre");
        String playlist = intent.getStringExtra("android.intent.extra.playlist");
        String title = intent.getStringExtra(MediaStore.EXTRA_MEDIA_TITLE);

        // Determine the search mode and use the corresponding extras
        if (mediaFocus == null) {
            // 'Unstructured' search mode (backward compatible)
            playUnstructuredSearch(query);

        } else if (mediaFocus.compareTo("vnd.android.cursor.item/*") == 0) {
            if (query.isEmpty()) {
                // 'Any' search mode
                playResumeLastPlaylist();
            } else {
                // 'Unstructured' search mode
                playUnstructuredSearch(query);
            }

        } else if (mediaFocus.compareTo(MediaStore.Audio.Genres.ENTRY_CONTENT_TYPE) == 0) {
            // 'Genre' search mode
            playGenre(genre);

        } else if (mediaFocus.compareTo(MediaStore.Audio.Artists.ENTRY_CONTENT_TYPE) == 0) {
            // 'Artist' search mode
            playArtist(artist, genre);

        } else if (mediaFocus.compareTo(MediaStore.Audio.Albums.ENTRY_CONTENT_TYPE) == 0) {
            // 'Album' search mode
            playAlbum(album, artist);

        } else if (mediaFocus.compareTo("vnd.android.cursor.item/audio") == 0) {
            // 'Song' search mode
            playSong(album, artist, genre, title);

        } else if (mediaFocus.compareTo(MediaStore.Audio.Playlists.ENTRY_CONTENT_TYPE) == 0) {
            // 'Playlist' search mode
            playPlaylist(album, artist, genre, playlist, title);
        }
    }
}
```

新笔记

创建笔记

如需创建新笔记，请使用 [ACTION_CREATE_NOTE](#) 操作并使用下文定义的 extra 指定笔记详情，例如主题和正文。

注：应用必须请求用户确认，然后才能完成操作。

操作

[ACTION_CREATE_NOTE](#)

数据 URI 架构

无

MIME 类型

PLAIN_TEXT_TYPE

/**/

Extra

EXTRA_NAME

一个表示笔记标题或主题的字符串。

EXTRA_TEXT

一个表示笔记正文的字符串。

示例 Intent：

```
public void createNote(String subject, String text) {
    Intent intent = new Intent(NoteIntents.ACTION_CREATE_NOTE)
        .putExtra(NoteIntents.EXTRA_NAME, subject)
        .putExtra(NoteIntents.EXTRA_TEXT, text);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="com.google.android.gms.actions.CREATE_NOTE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="*/*">
    </intent-filter>
</activity>
```

电话

发起通话

如需打开电话应用并拨打电话号码，请使用 ACTION_DIAL 操作，并使用下文定义的 URI 架构指定电话号码。电话应用打开时会显示电话号码，但用户必需按拨打电话按钮才能开始通话。

如需直接拨打电话，请使用 ACTION_CALL 操作，并使用下文定义的 URI 架构指定电话号码。电话应用打开时便会拨打电话，用户无需按拨打电话按钮。


ACTION_CALL 操作需要您在清单文件中添加 CALL_PHONE 权限：

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

操作

- ACTION_DIAL - 打开拨号器或电话应用。
- ACTION_CALL - 拨打电话（需要 CALL_PHONE 权限）

数据 URI 架构

 Google Voice Actions

- “致电 555-5555”
- “致电 bob”
- “致电语音邮件”

- `tel:<phone-number>`
- `voicemail:<phone-number>`

MIME 类型

无

有效电话号码是指符合 [IETF RFC 3966](#) 规定的号码。举例来说，有效电话号码包括下列号码：

- `tel:2125551212`
- `tel:(212) 555 1212`

电话的拨号器能够很好地对架构进行标准化，如电话号码。因此并不严格要求 `Uri.parse()` 方法中必须使用所述架构。不过，如果您尚未试用过架构，或者不确定是否可以处理架构，请改用 `Uri.fromParts()` 方法。

示例 Intent：

```
public void dialPhoneNumber(String phoneNumber) {
    Intent intent = new Intent(Intent.ACTION_DIAL);
    intent.setData(Uri.parse("tel:" + phoneNumber));
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

搜索

使用特定应用搜索



Google Voice Actions

- “在我的视频应用中搜索有关猫的视频”



视频
在您的应用中进行语音搜索

如需支持在您的应用环境内进行搜索，请使用 `SEARCH_ACTION` 操作在您的应用中声明一个 Intent 过滤器，如下文示例 Intent 过滤器中所示。

操作

`"com.google.android.gms.actions.SEARCH_ACTION"`

支持来自 Google Voice Actions 的搜索查询。

Extra

`QUERY`

一个包含搜索查询的字符串。

示例 Intent 过滤器：


```
<activity android:name=".SearchActivity">
    <intent-filter>
        <action android:name="com.google.android.gms.actions.SEARCH_ACTION"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

执行网页搜索

如需发起网页搜索，请使用 `ACTION_WEB_SEARCH` 操作，并在 `SearchManager.QUERY` extra 中指定搜索字符串。

操作

`ACTION_WEB_SEARCH`

数据 URI 架构

无

MIME 类型

无

Extra

`SearchManager.QUERY`

搜索字符串。

示例 Intent：

```
public void searchWeb(String query) {
    Intent intent = new Intent(Intent.ACTION_SEARCH);
    intent.putExtra(SearchManager.QUERY, query);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

设置

打开特定设置部分

如需在您的应用要求用户更改内容时打开某个系统设置屏幕，请使用下列其中一个 Intent 操作打开与操作名称对应的设置屏幕。

操作

`ACTION_SETTINGS`

`ACTION_WIRELESS_SETTINGS`

`ACTION_AIRPLANE_MODE_SETTINGS`

`ACTION_WIFI_SETTINGS`

`ACTION_APN_SETTINGS`

`ACTION_BLUETOOTH_SETTINGS`

`ACTION_DATE_SETTINGS`

`ACTION_LOCALE_SETTINGS`

`ACTION_INPUT_METHOD_SETTINGS`

`ACTION_DISPLAY_SETTINGS`

`ACTION_SECURITY_SETTINGS`

[ACTION_LOCATION_SOURCE_SETTINGS](#)
[ACTION_INTERNAL_STORAGE_SETTINGS](#)
[ACTION_MEMORY_CARD_SETTINGS](#)

有关其他可用的设置屏幕，请参见 [Settings](#) 文档。

数据 URI 架构

无

MIME 类型

无

示例 Intent：

```
public void openWifiSettings() {  
    Intent intent = new Intent(Intent.ACTION_WIFI_SETTINGS);  
    if (intent.resolveActivity(getPackageManager()) != null) {  
        startActivity(intent);  
    }  
}
```

发送短信

撰写带附件的短信/彩信

如需发起短信或彩信，请使用以下其中一个 Intent 操作，并使用下列 extra 键指定电话号码、主题和消息正文等消息详情。

操作

[ACTION_SENDTO](#) 或
[ACTION_SEND](#) 或
[ACTION_SEND_MULTIPLE](#)

数据 URI 架构

sms:<phone_number>

smsto:<phone_number>

mms:<phone_number>

mmsto:<phone_number>

以上每一个架构的处理方式都相同。

MIME 类型

"text/plain"

"image/*"

"video/*"

Extra

"subject"

表示消息主题的字符串（通常只适用于彩信）。

"sms_body"

表示消息正文的字符串。

EXTRA_STREAM

指向要附加的图像或视频的 [Uri](#)。如果使用的是 [ACTION_SEND_MULTIPLE](#) 操作，此 extra 应为指向要附加的图像/视频的 [Uri ArrayList](#)。

示例 Intent：

```
public void composeMmsMessage(String message, Uri attachment) {
    Intent intent = new Intent(Intent.ACTION_SENDTO);
    intent.setType(HTTP.PLAIN_TEXT_TYPE);
    intent.putExtra("sms_body", message);
    intent.putExtra(Intent.EXTRA_STREAM, attachment);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

如果您想确保 Intent 只由短信应用（而非其他电子邮件或社交应用）进行处理，则需使用 [ACTION_SENDTO](#) 操作并加入 "smsto:" 数据架构。例如：

```
public void composeMmsMessage(String message, Uri attachment) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setData(Uri.parse("smsto:")); // This ensures only SMS apps respond
    intent.putExtra("sms_body", message);
    intent.putExtra(Intent.EXTRA_STREAM, attachment);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <data android:type="text/plain" />
        <data android:type="image/*" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

注：如果您要开发短信/彩信应用，必须为几项额外操作实现 Intent 过滤器，才能在 Android 4.4 及更高版本上成为默认短信应用。如需了解详细信息，请参见 [Telephony](#) 处的文档。

网络浏览器

加载网址

如需打开网页，请使用 [ACTION_VIEW](#) 操作，并在 Intent 数据中指定网址。

操作

[ACTION_VIEW](#)



Google Voice Actions

- “打开 example.com”

数据 URI 架构

```
http:<URL>
https:<URL>
```

MIME 类型

```
"text/plain"

"text/html"

"application/xhtml+xml"

"application/vnd.wap.xhtml+xml"
```

示例 Intent：

```
public void openWebPage(String url) {
    Uri webpage = Uri.parse(url);
    Intent intent = new Intent(Intent.ACTION_VIEW, webpage);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

示例 Intent 过滤器：

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <!-- Include the host attribute if you want your app to respond
             only to URLs with your app's domain. -->
        <data android:scheme="http" android:host="www.example.com" />
        <category android:name="android.intent.category.DEFAULT" />
        <!-- The BROWSABLE category is required to get links from web pages. -->
        <category android:name="android.intent.category.BROWSABLE" />
    </intent-filter>
</activity>
```

提示：如果您的 Android 应用提供与您的网站相似的功能，请为指向您的网站的 URL 加入一个 Intent 过滤器。之后，如果用户安装了您的应用，点击电子邮件或其他网页中指向您的网站的链接时，将会打开您的 Android 应用而不是您的网页。

使用 Android 调试桥验证 Intent

如需验证您的应用可以对您想支持的 Intent 作出响应，可以使用 `adb` 工具来触发特定 Intent：

1. 设置一台用于开发的 Android 设备，或使用一台虚拟设备。
2. 安装一个处理您想支持的 Intent 的应用版本。
3. 使用 `adb` 触发一个 Intent：

```
adb shell am start -a <ACTION> -t <MIME_TYPE> -d <DATA> \
    -e <EXTRA_NAME> <EXTRA_VALUE> -n <ACTIVITY>
```

例如：

```
adb shell am start -a android.intent.action.DIAL \
    -d tel:555-5555 -n org.example.MyApp/.MyActivity
```

4. 如果您定义了必需的 Intent 过滤器，您的应用应该会处理 Intent。

如需了解详细信息，请参见 [ADB Shell 命令](#)。