联系人提供程序

内容快览

- Android 的联系人相关信息存储区。
- 与网页同步。
- 集成社交流数据。

本文内容

- 联系人提供程序组织
- 原始联系人
- 数据
- 联系人
- 来自同步适配器的数据
- 所需权限
- 用户个人资料
- 联系人提供程序元数据
- 联系人提供程序访问
- 联系人提供程序同步适配器
- 社交流数据
- 其他联系人提供程序功能

关键类

- `ContactsContract.Contacts`
- `ContactsContract.RawContacts`
- `ContactsContract.Data`
- android.provider.ContactsContract.StreamItems

相关示例

- 联系人管理器
- 示例同步适配器

另请参阅

- 内容提供程序基础知识

联系人提供程序是一个强大而又灵活的 Android 组件，用于管理设备上联系人相关数据的中央存储区。 联系人提供程序是您在设备的联系人应用中看到的数据源，您也可以在自己的应用中访问其数据，并可在设备与在线服务之间传送数据。 提供程序储存有多种数据源，由于它会试图为每个联系人管理尽可能多的数据，因此造成其组织结构非常复杂。 为此，该提供程序的 API 包含丰富的协定类和接口，为数据检索和修改提供便利。

本指南介绍下列内容：

- 提供程序基本结构
- 如何从提供程序检索数据
- 如何修改提供程序中的数据
- 如何编写用于同步服务器数据与联系人提供程序数据的同步适配器。

本指南假定您了解 Android 内容提供程序的基础知识。如需了解有关 Android 内容提供程序的更多信息，请阅读 内容提供程序基础知识指南。 示例同步适配器示例应用是一个示例，展示如何使用同步适配器在联系人提供程序与 Google 网络服务托管的一个示例应用之间传送数据。

联系人提供程序组织

联系人提供程序是 Android 内容提供程序的一个组件。它保留了三种类型的联系人数据，每一种数据都对应提供程序提供的一个表，如图 1 所示：

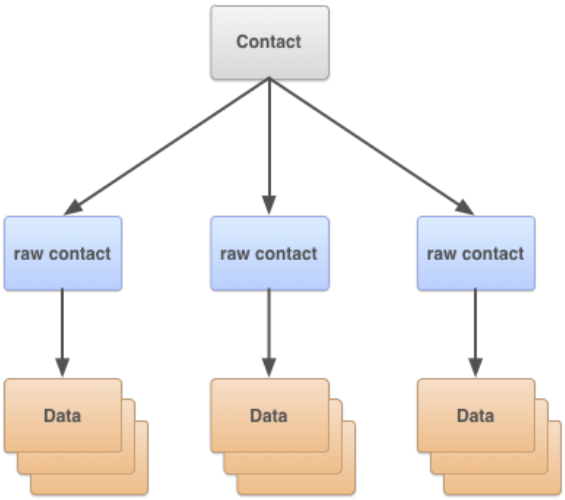

图 1. 联系人提供程序表结构。

这三个表通常以其协定类的名称命名。这些类定义表所使用的内容 URI、列名称及列值相应的常量：

`ContactsContract.Contacts` 表

表示不同联系人的行，基于聚合的原始联系人行。

`ContactsContract.RawContacts` 表

包含联系人数据摘要的行，针对特定用户帐户和类型。

`ContactsContract.Data` 表

包含原始联系人详细信息（例如电子邮件地址或电话号码）的行。

由 `ContactsContract` 中的协定类表示的其他表是辅助表，联系人提供程序利用它们来管理其操作，或为设备的联系人或电话应用中的特定功能提供支持。

原始联系人

一个原始联系人表示来自某一帐户类型和帐户名称、有关某个联系人的数据。 由于联系人提供程序允许将多个在线服务作为某一联系人的数据源，因此它允许同一联系人对应多个原始联系人。 借助支持多个原始联系人的特性，用户还可以将某一联系人在帐户类型相同的多个帐户中的数据进行合并。

原始联系人的大部分数据并不存储在 `ContactsContract.RawContacts` 表内，而是存储在 `ContactsContract.Data` 表中的一行或多行内。每个数据行都有一个 `Data.RAW_CONTACT_ID` 列，其中包含其父级 `ContactsContract.RawContacts` 行的 `RawContacts._ID` 值。

重要的原始联系人列

表 1 列出了 `ContactsContract.RawContacts` 表中的重要列。请阅读表后的说明：

表 1. 重要的原始联系人列。

列名称	用途	说明
`ACCOUNT_NAME`	作为该原始联系人来源的帐户类型的帐户名称。 例如，Google 帐户的帐户名称是设备所有者的某个 Gmail 地址。如需了解详细信息，请参阅有关 `ACCOUNT_TYPE` 的下一条目。	此名称的格式专用于其帐户类型。它不一定是电子邮件地址。
`ACCOUNT_TYPE`	作为该原始联系人来源的帐户类型。例如，Google 帐户的帐户类型是 `com.google`。 请务必使用您拥有或控制的域的域标识符限定您的帐户类型。 这可以确保您的帐户类型具有唯一性。	提供联系人数据的帐户类型通常关联有同步适配器，用于与联系人提供程序进行同步。
`DELETED`	原始联系人的“已删除”标志。	此标志让联系人提供程序能够在内部保留该行，直至同步适配器能够从服务器删除该行，然后再从存储区中最终删除该行。

说明

以下是关于 ContactsContract.RawContacts 表的重要说明：

- 原始联系人的姓名并不存储其在 ContactsContract.RawContacts 中的行内，而是存储在 ContactsContract.Data 表的 ContactsContract.CommonDataKinds.StructuredName 行内。一个原始联系人在 ContactsContract.Data 表中只有一个该类型的行。
- **注意：**要想在原始联系人行中使用您自己的帐户数据，必须先在 AccountManager 中注册帐户。 为此，请提示用户将帐户类型及其帐户名称添加到帐户列表。 如果您不这样做，联系人提供程序将自动删除您的原始联系人行。

  例如，如果您想让您的应用为您域名为 com.example.dataservice、基于网络的服务保留联系人数据，并且您的服务的用户帐户是 becky.sharp@dataservice.example.com，则用户必须先添加帐户“类型”(com.example.dataservice) 和帐户“名称”(becky.smart@dataservice.example.com)，然后您的应用才能添加原始联系人行。 您可以在文档中向用户解释这项要求，也可以提示用户添加类型和名称，或者同时采用这两种措施。 下文对帐户类型和帐户名称做了更详尽的描述。

原始联系人数据来源

为理解原始联系人的工作方式，假设有一位用户“Emily Dickinson”，她的设备上定义了以下三个用户帐户：

- emily.dickinson@gmail.com
- emilyd@gmail.com
- Twitter 帐户“belle_of_amherst”

该用户已在“Accounts”设置中为全部三个帐户启用了“Sync Contacts”。

假定 Emily Dickinson 打开一个浏览器窗口，以 emily.dickinson@gmail.com 身份登录 Gmail，然后打开 “联系人”，并添加“Thomas Higginson”。后来，她以 emilyd@gmail.com 身份登录 Gmail，并向“Thomas Higginson”发送一封电子邮件，此操作会自动将他添加为联系人。 她还在 Twitter 上关注了“colonel_tom”（Thomas Higginson 的 Twitter ID）。

以上操作的结果是，联系人提供程序会创建以下这三个原始联系人：

1. 第一个原始联系人对应“Thomas Higginson”，关联帐户 emily.dickinson@gmail.com。 用户帐户类型是 Google。
2. 第二个原始联系人对应“Thomas Higginson”，关联帐户 emilyd@gmail.com。 用户帐户类型也是 Google。由于添加的联系人对应的用户帐户不同，因此尽管名称与前一名称完全相同，也只能作为第二个原始联系人。
3. 第三个原始联系人对应“Thomas Higginson”，关联帐户“belle_of_amherst”。用户帐户类型是 Twitter。

数据

所前所述，原始联系人的数据存储在一个 ContactsContract.Data 行中，该行链接到原始联系人的 _ID 值。这使一位原始联系人可以拥有多个具有相同数据类型的实例，例如电子邮件地址或电话号码。 例如，如果对应 emilyd@gmail.com 的“Thomas Higginson”（关联 Google 帐户 emilyd@gmail.com 的 Thomas Higginson 的原始联系人行）的住宅电子邮件地址为 thigg@gmail.com，办公电子邮件地址为 thomas.higginson@gmail.com，则联系人提供程序会存储这两个电子邮件地址行，并将它们都链接到原始联系人。

请注意，这个表中存储了不同类型的数据。显示姓名、电话号码、电子邮件、邮政地址、照片以及网站明细行都可以在 ContactsContract.Data 表中找到。 为便于管理这些数据， ContactsContract.Data 表为一些列使用了描述性名称，为其他列使用了通用名称。 使用描述性名称的列的内容具有相同的含义，与行中数据的类型无关，而使用通用名称的列的内容则会随数据类型的不同而具有不同的含义。

描述性列名称

以下是一些描述性列名称的示例：

RAW_CONTACT_ID

> 该数据对应的原始联系人 _ID 列的值。

MIMETYPE

> 该行中存储的数据类型，以自定义 MIME（多用途互联网邮件扩展）类型表示。联系人提供程序使用了 ContactsContract.CommonDataKinds 子类中定义的 MIME 类型。 这些 MIME 类型为开源类型，可供与联系人提供程序协作的任何应用或同步适配器使用。

IS_PRIMARY

> 如果一个原始联系人可能具有多个这种类型的数据行， IS_PRIMARY 列会标记 包含该类型主要数据的数据行。例如，如果用户长按某个联系人的电话号码，并选择 **Set default**，则包含该号码的 ContactsContract.Data 行会将其 IS_PRIMARY 列设置为一个非零值。

通用列名称

有 15 个通用列命名为 DATA1 至 DATA15，可普遍适用；还有四个通用列命名为 SYNC1 至 SYNC4，只应由同步适配器使用。 通用列名称常量始终有效，与行包含的数据类型无关。

DATA1 列为索引列。联系人提供程序总是在此列中存储其预期会成为最频繁查询目标的数据。 例如，在一个电子邮件行中，此列包含实际电子邮件地址。

按照惯例，DATA15 为预留列，用于存储照片缩略图等二进制大型对象 (BLOB) 数据。

类型专用列名称

为便于处理特定类型行的列，联系人提供程序还提供了 ContactsContract.CommonDataKinds 子类中定义的类型专用列名称常量。 这些常量只是为同一列名称提供不同的常量名称，这有助于您访问特定类型行中的数据。

例如，ContactsContract.CommonDataKinds.Email 类为 ContactsContract.Data 行定义类型专用列名称常量，该行的 MIME 类型为 Email.CONTENT_ITEM_TYPE。 该类包含电子邮件地址列的 ADDRESS 常量。 ADDRESS 的实际值为“data1”，这与列的通用名称相同。

> **注意：**请勿使用具有提供程序某个预定义 MIME 类型的行向 ContactsContract.Data 表中添加您自己的自定义数据。 否则您可能会丢失数据，或导致提供程序发生故障。 例如，如果某一行具有 MIME 类型 Email.CONTENT_ITEM_TYPE，并且 DATA1 列包含的是用户名而不是电子邮件地址，您就不应添加该行。如果您为该行使用自定义的 MIME 类型，则可自由定义您的自定义类型专用的列名称，并随心所欲地使用这些列。

图 2 显示的是描述性列和数据列在 ContactsContract.Data 行中的显示情况，以及类型专用列名称“覆盖”通用列名称的情况

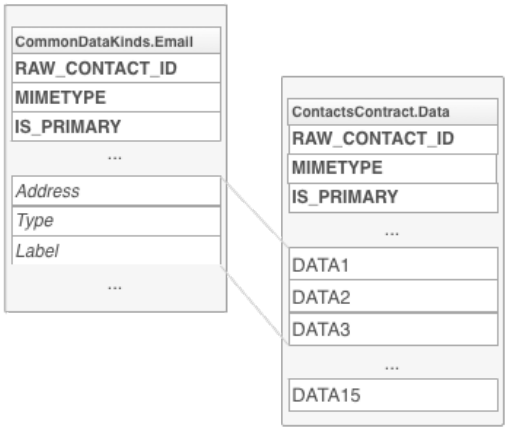

图 2. 类型专用列名称和通用列名称。

类型专用列名称类

表 2 列出了最常用的类型专用列名称类：

表 2. 类型专用列名称类

映射类	数据类型	说明
ContactsContract.CommonDataKinds.StructuredName	与该数据行关联的原始联系人的姓名数据。	一位原始联系人只有其中一行。
ContactsContract.CommonDataKinds.Photo	与该数据行关联的原始联系人的主要照片。	一位原始联系人只有其中一行。
ContactsContract.CommonDataKinds.Email	与该数据行关联的原始联系人的电子邮件地址。	一位原始联系人可有多个电子邮件地址。
ContactsContract.CommonDataKinds.StructuredPostal	与该数据行关联的原始联系人的邮政地址。	一位原始联系人可有多个邮政地址。
ContactsContract.CommonDataKinds.GroupMembership	将原始联系人链接到联系人提供程序内其中一组的标识符。	组是帐户类型和帐户名称的一项可选功能。联系人组部分对其做了更详尽的描述。

联系人

联系人提供程序通过将所有帐户类型和帐户名称的原始联系人行合并来形成**联系人**。 这可以为显示和修改用户针对某一联系人收集的所有数据提供便利。 联系人提供程序管理新联系人行的创建，以及原始联系人与现有联系人行的合并。 系统不允许应用或同步适配器添加联系人，并且联系人行中的某些列是只读列。

> **注：**如果您试图通过 `insert()` 向联系人提供程序添加联系人，会引发一个 `UnsupportedOperationException` 异常。 如果您试图更新一个列为“只读”的列，更新会被忽略。

如果添加的新原始联系人不匹配任何现有联系人，联系人提供程序会相应地创建新联系人。 如果某个现有原始联系人的数据发生了变化，不再匹配其之前关联的联系人，则提供程序也会执行此操作。 如果应用或同步适配器创建的新原始联系人*的确*匹配某位现有联系人，则新原始联系人将与现有联系人合并。

联系人提供程序通过 `Contacts` 表中联系人行的 `_ID` 列将联系人行与其各原始联系人行链接起来。 原始联系人表 `ContactsContract.RawContacts` 的 `CONTACT_ID` 列包含对应于每个原始联系人行所关联联系人行的 `_ID` 值。

`ContactsContract.Contacts` 表还有一个 `LOOKUP_KEY` 列，它是一个指向联系人行的“永久性”链接。 由于联系人提供程序会自动维护联系人，因此可能会在合并或同步时相应地更改联系人行的 `_ID` 值。 即使发生这种情况，合并了联系人 `LOOKUP_KEY` 的内容 URI `CONTENT_LOOKUP_URI` 仍将指向联系人行，这样，您就能使用 `LOOKUP_KEY` 保持指向“最喜爱”联系人的链接，以及执行其他操作。 该列具有其自己的格式，与 `_ID` 列的格式无关。

图 3 显示的是这三个主要表的相互关系。

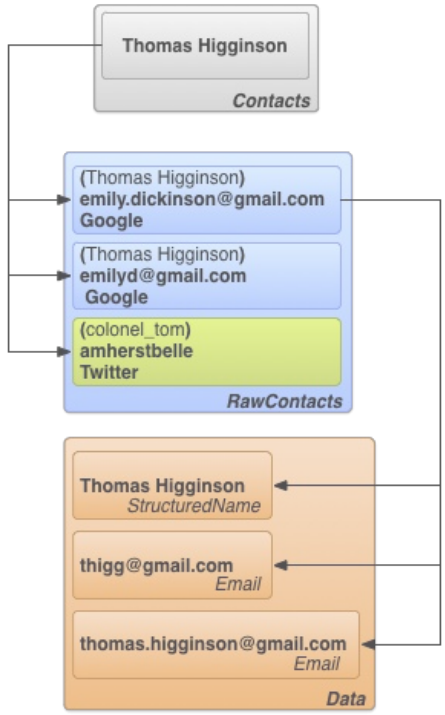

图 3. 联系人表、原始联系人表与详细信息表之间的关系。

来自同步适配器的数据

虽然用户是直接将联系人数据输入到设备中，但这些数据也会通过**同步适配器**从网络服务流入联系人提供程序中，这些同步适配器可自动化设备与服务之间的数据传送。 同步适配器在系统控制下在后台运行，它们会调用 `ContentResolver` 方法来管理数据。

在 Android 中，与同步适配器协作的网络服务通过帐户类型加以标识。 每个同步适配器都与一个帐户类型协作，但它可以支持该类型的多个帐户名称。 原始联系人数据来源部分对帐户类型和帐户名称做了简要描述。 下列定义提供了更多详细信息，并描述了帐户类型及帐户名称与同步适配器及服务之间的关系。

帐户类型

> 表示用户在其中存储数据的服务。在大多数时候，用户需要向服务验证身份。 例如，Google 通讯录是一个以代码 `google.com` 标识的帐户类型。该值对应于 `AccountManager` 使用的帐户类型。

帐户名称

> 表示某个帐户类型的特定帐户或登录名。Google 通讯录帐户与 Google 帐户相同，都是以电子邮件地址作为帐户名称。 其他服务可能使用一个单词的用户名或数字 ID。

帐户类型不必具有唯一性。用户可以配置多个 Google 通讯录帐户并将它们的数据下载到联系人提供程序；如果用户为个人帐户名称和工作帐户名称分别设置了一组联系人，就可能发生这种情况。 帐户名称通常具有唯一性。 它们共同标识联系人提供程序与外部服务之间的特定数据流。

如果您想将服务的数据传送到联系人提供程序，则需编写您自己的同步适配器。 联系人提供程序同步适配器部分对此做了更详尽的描述。

图 4 显示的是联系人提供程序如何融入联系人数据的流动。 在名为“同步适配器”的方框中，每个适配器都以其帐户类型命名。

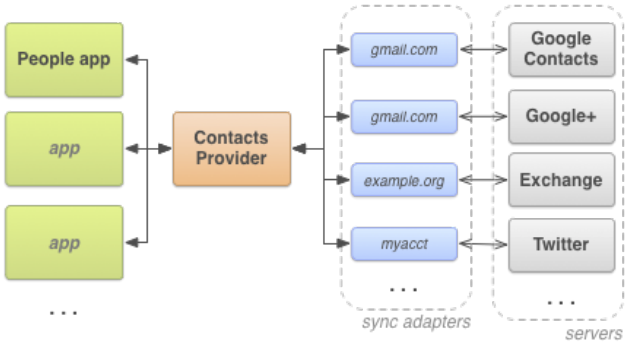

图 4. 联系人提供程序数据流。

所需权限

想要访问联系人提供程序的应用必须请求以下权限：

对一个或多个表的读取权限

> `READ_CONTACTS`，在 `AndroidManifest.xml` 中指定，使用 `<uses-permission>` 元素作为 `<uses-permission android:name="android.permission.READ_CONTACTS">`。

对一个或多个表的写入权限

> `WRITE_CONTACTS`，在 `AndroidManifest.xml` 中指定，使用 `<uses-permission>` 元素作为 `<uses-permission android:name="android.permission.WRITE_CONTACTS">`。

这些权限不适用于用户个人资料数据。下面的用户个人资料部分对用户个人资料及其所需权限做了阐述。

请切记，用户的联系人数据属于个人敏感数据。用户关心其隐私权，因此不希望应用收集有关其自身的数据或其联系人的数据。 如需权限来访问其联系人数据的理由并不充分，用户可能给您的应用作出差评或干脆拒绝安装。

用户个人资料

`ContactsContract.Contacts` 表有一行包含设备用户的个人资料数据。 这些数据描述设备的 `user` 而不是用户的其中一位联系人。 对于每个使用个人资料的系统，该个人资料联系人行都链接到某个原始联系人行。 每个个人资料原始联系人行可具有多个数据行。`ContactsContract.Profile` 类中提供了用于访问用户个人资料的常量。

访问用户个人资料需要特殊权限。除了进行读取和写入所需的 `READ_CONTACTS` 和 `WRITE_CONTACTS` 权限外，如果想访问用户个人资料，还分别需要 android.Manifest.permission#READ_PROFILE 和 android.Manifest.permission#WRITE_PROFILE 权限进行读取和写入访问。

请切记，您应该将用户的个人资料视为敏感数据。权限 android.Manifest.permission#READ_PROFILE 让您可以访问设备用户的个人身份识别数据。 请务必在您的应用的描述中告知用户您需要用户个人资料访问权限的原因。

要检索包含用户个人资料的联系人行，请调用 `ContentResolver.query()`。 将内容 URI 设置为 `CONTENT_URI` 并且不要提供任何选择条件。 您还可以使用该内容 URI 作为检索原始联系人或个人资料数据的基本 URI。 例如，以下代码段用于检索个人资料数据：

```
// Sets the columns to retrieve for the user profile
mProjection = new String[]
    {
        Profile._ID,
        Profile.DISPLAY_NAME_PRIMARY,
        Profile.LOOKUP_KEY,
        Profile.PHOTO_THUMBNAIL_URI
    };

// Retrieves the profile from the Contacts Provider
mProfileCursor =
        getContentResolver().query(
                Profile.CONTENT_URI,
                mProjection ,
                null,
                null,
                null);
```

> **注：**如果您要检索多个联系人行并想要确定其中一个是否为用户个人资料，请测试该行的 `IS_USER_PROFILE` 列。 如果该联系人是用户个人资料，则此列设置为“1”。

联系人提供程序元数据

联系人提供程序管理用于追踪存储区中联系人数据状态的数据。 这些有关存储区的元数据存储在各处，其中包括原始联系人表行、数据表行和联系人表行、 `ContactsContract.Settings` 表以及 `ContactsContract.SyncState` 表。 下表显示的是每一部分元数据的作用：

表 3. 联系人提供程序中的元数据

表	列	值	含义
`ContactsContract.RawContacts`	`DIRTY`	“0”：上次同步以来未发生变化。	标记设备上因发生变化而需要同步回服务器的原始联系人。 当 Android 应用更新行时，联系人提供程序会自动设置该值。 修改原始联系人表或数据表的同步适配器应始终向他们使用的内容 URI 追加字符串 `CALLER_IS_SYNCADAPTER`。 这可以防止提供程序将行标记为已更新。 否则，即使服务器是修改的来源，同步适配器修改仍显示为本地修改，并会发送到服务器。
		“1”：上次同步以来发生了变化，需要同步回服务器。	
`ContactsContract.RawContacts`	`VERSION`	此行的版本号。	每当行或其相关数据发生变化时，联系人提供程序都会自动增加此值。
`ContactsContract.Data`	`DATA_VERSION`	此行的版本号。	每当数据行发生变化时，联系人提供程序都会自动增加此值。
`ContactsContract.RawContacts`	`SOURCE_ID`	一个字符串值，用于在创建此原始联系人的帐户中对该联系人进行唯一标识。	当同步适配器创建新原始联系人时，此列应设置为该原始联系人在服务器中的唯一 ID。 当 Android 应用创建新原始联系人时，应将此列留空。 这是为了向同步适配器表明，它应该在服务器上创建新原始联系人，并获取 `SOURCE_ID` 的值。 具体地讲，对于每个帐户类型，该源 ID 都必须是**唯一**的，并且应在所有同步中保持稳定： • 唯一：帐户的每个原始联系人都必须有自己的源 ID。如果您不强制执行此要求，会在联系人应用中引发问题。 请注意，帐户*类型*相同的两个原始联系人可以具有相同的源 ID。 例如，允许帐户 `emily.dickinson@gmail.com` 的原始联系人“Thomas Higginson”与帐户 `emilyd@gmail.com` 的原始联系人“Thomas Higginson”具有相同的源 ID。 • 稳定：源 ID 是该原始联系人在在线服务中的数据的永久性组成部分。 例如，如果用户从应用设置中清除存储的联系人数据并重新同步，则恢复的原始联系人的源 ID 应与以前相同。 如果您不强制执行此要求，快捷方式将停止工作。
`ContactsContract.Groups`	`GROUP_VISIBLE`	“0”：此组中的联系人在 Android 应用 UI 中不应处于可见状态。	此列用于兼容那些允许用户隐藏特定组中联系人的服务器。
		“1”：系统允许此组中的联系人在应用 UI 中处于可见状态。	
`ContactsContract.Settings`	`UNGROUPED_VISIBLE`	“0”：对于此帐户和帐户类型，未归入组的联系人在 Android 应用 UI 中处	默认情况下，如果联系人的所有原始联系人都未归入组，则它们将处于不可见状态（原始联系人的组成员身份通过 `ContactsContract.Data` 表中的一个或多个 `ContactsContract.CommonDataKinds.GroupMembership` 行指示）。 通过在 `ContactsContract.Settings` 表行中为帐户类型和帐户设置此标志，您可以强制未归入组的联系人处于可见状态。 此标志的一个用途是显示不使用组的服务器上的联系人。

		于不可见状态。 “1”：对于此帐户和帐户类型，未归入组的联系人在应用 UI 中处于可见状态。	
`ContactsContract.SyncState`	(all)	此表用于存储同步适配器的元数据。	利用此表，您可以将同步状态及其他同步相关数据持久地存储在设备中。

联系人提供程序访问

本节描述访问联系人提供程序中数据的准则，侧重于阐述以下内容：

- 实体查询。
- 批量修改。
- 通过 Intent 执行检索和修改。
- 数据完整性。

联系人提供程序同步适配器部分也对通过同步适配器进行修改做了更详尽的阐述。

查询实体

由于联系人提供程序表是以层级形式组织，因此对于检索某一行以及与其链接的所有“子”行，往往很有帮助。 例如，要想显示某位联系人的所有信息，您可能需要检索某个 `ContactsContract.Contacts` 行的所有`ContactsContract.RawContacts` 行，或者检索某个 `ContactsContract.RawContacts` 行的所有 `ContactsContract.CommonDataKinds.Email` 行。 为便于执行此操作，联系人提供程序提供了**实体**构造，其作用类似于表间的数据库连接。

实体类似于一个表，由父表及其子表中的选定列组成。 当您查询实体时，需要根据实体中的可用列提供投影和搜索条件。 结果会得到一个 `Cursor`，检索的每个子表行在其中都有一行与之对应。 例如，如果您在 `ContactsContract.Contacts.Entity` 中查询某个联系人姓名以及该姓名所有原始联系人的所有 `ContactsContract.CommonDataKinds.Email` 行，您会获得一个 `Cursor`，每个 `ContactsContract.CommonDataKinds.Email` 行在其中都有一行与之对应。

实体简化了查询。使用实体时，您可以一次性检索联系人或原始联系人的所有联系人数据，而不必先通过查询父表获得 ID，然后通过该 ID 查询子表。 此外，联系人提供程序可通过单一事务处理实体查询，这确保了所检索数据的内部一致性。

> **注：**实体通常不包含父表和子表的所有列。 如果您试图使用的列名称并未出现在实体的列名称常量列表中，则会引发一个 `Exception`。

以下代码段说明如何检索某位联系人的所有原始联系人行。该代码段是一个大型应用的组成部分，包含“主”和“详”两个 Activity。 主 Activity 显示一个联系人行列表；当用户选择一行时，该 Activity 会将其 ID 发送至详 Activity。 详 Activity 使用 `ContactsContract.Contacts.Entity` 显示与所选联系人关联的所有原始联系人中的所有数据行。

以下代码段摘自“详”Activity：

```
...
    /*
     * Appends the entity path to the URI. In the case of the Contacts Provider, the
     * expected URI is content://com.google.contacts/#/entity (# is the ID value).
     */
    mContactUri = Uri.withAppendedPath(
            mContactUri,
            ContactsContract.Contacts.Entity.CONTENT_DIRECTORY);

    // Initializes the loader identified by LOADER_ID.
    getLoaderManager().initLoader(
            LOADER_ID,  // The identifier of the loader to initialize
            null,       // Arguments for the loader (in this case, none)
            this);      // The context of the activity

    // Creates a new cursor adapter to attach to the list view
    mCursorAdapter = new SimpleCursorAdapter(
            this,                        // the context of the activity
            R.layout.detail_list_item,   // the view item containing the detail widgets
            mCursor,                     // the backing cursor
            mFromColumns,                // the columns in the cursor that provide the data
            mToViews,                    // the views in the view item that display the data
            0);                          // flags

    // Sets the ListView's backing adapter.
    mRawContactList.setAdapter(mCursorAdapter);
...
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {

    /*
     * Sets the columns to retrieve.
     * RAW_CONTACT_ID is included to identify the raw contact associated with the data row.
     * DATA1 contains the first column in the data row (usually the most important one).
     * MIMETYPE indicates the type of data in the data row.
     */
    String[] projection =
        {
            ContactsContract.Contacts.Entity.RAW_CONTACT_ID,
            ContactsContract.Contacts.Entity.DATA1,
            ContactsContract.Contacts.Entity.MIMETYPE
        };

    /*
     * Sorts the retrieved cursor by raw contact id, to keep all data rows for a single raw
     * contact collated together.
     */
    String sortOrder =
            ContactsContract.Contacts.Entity.RAW_CONTACT_ID +
            " ASC";

    /*
     * Returns a new CursorLoader. The arguments are similar to
     * ContentResolver.query(), except for the Context argument, which supplies the location of
     * the ContentResolver to use.
     */
    return new CursorLoader(
            getApplicationContext(),  // The activity's context
            mContactUri,              // The entity content URI for a single contact
            projection,               // The columns to retrieve
            null,                     // Retrieve all the raw contacts and their data rows.
            null,                     //
            sortOrder);               // Sort by the raw contact ID.
}
```

加载完成时，`LoaderManager` 会调用一个 `onLoadFinished()` 回调。此方法的传入参数之一是一个 `Cursor`，其中包含查询的结果。在您自己的应用中，您可以从该 `Cursor` 获取数据，以进行显示或做进一步处理。

批量修改

您应尽可能地通过创建一个 `ContentProviderOperation` 对象 `ArrayList` 并调用 `applyBatch()`，以“批处理模式”在联系人提供程序中插入、更新和删除数据。 由于联系人提供程序是在 `applyBatch()` 中通过单一事务执行所有操作，因此您的修改绝不会使联系人存储区出现不一致问题。 此外，批量修改还有便于同时插入原始联系人及其明细数据。

> **注：**要修改*单个*原始联系人，可以考虑向设备的联系人应用发送一个 Intent，而不是在您的应用中处理修改。 通过 Intent 执行检索和修改部分对此操

> 作做了更详尽的描述。

屈服点

一个包含大量操作的批量修改可能会阻断其他进程，导致糟糕的总体用户体验。 要将您想执行的所有修改组织到尽可能少的单独列表中，同时防止它们阻断系统，则应为一项或多项操作设置**屈服点**。 屈服点是一个 `ContentProviderOperation` 对象，其 `isYieldAllowed()` 值设置为 `true`。当联系人提供程序遇到屈服点时，它会暂停其工作，让其他进程运行，并关闭当前事务。 当提供程序再次启动时，它会继续执行 `ArrayList` 中的下一项操作，并启动一个新的事务。

屈服点会导致每次调用 `applyBatch()` 会产生多个事务。因此，您应该为针对一组相关行的最后一项操作设置屈服点。 例如，您应该为一组操作中添加原始联系人行及其关联数据行的最后一项操作，或者针对一组与一位联系人相关的行的最后一项操作设置屈服点。

屈服点也是一个原子操作单元。两个屈服点之间所有访问的成功或失败都将以一个单元的形式出现。 如果您不设置任何屈服点，则最小的原子操作是整个批量操作。 如果您使用了屈服点，则可以防止操作降低系统性能，还可确保一部分操作是原子操作。

修改向后引用

当您将一个新原始联系人行及其关联的数据行作为一组 `ContentProviderOperation` 对象插入时，需要通过将原始联系人的 `_ID` 值作为 `RAW_CONTACT_ID` 值插入，将数据行链接到原始联系人行。 不过，当您为数据行创建 `ContentProviderOperation` 时，该值不可用，因为您尚未对原始联系人行应用 `ContentProviderOperation`。 为解决此问题， `ContentProviderOperation.Builder` 类使用了 `withValueBackReference()` 方法。 该方法让您可以插入或修改包含上一操作结果的列。

`withValueBackReference()` 方法具有两个参数：

`key`

> 键-值对的键。此参数的值应为您要修改的表中某一列的名称。

`previousResult`

> `applyBatch()` 中 `ContentProviderResult` 对象数组内某一值以 0 开始的索引。 应用批处理操作时，每个操作的结果都存储在一个中间结果数组内。 `previousResult` 值是其中一个结果的索引，它通过 `key` 值进行检索和存储。 这样，您就可以插入一条新的原始联系人记录，并取回其 `_ID` 值，然后在添加 `ContactsContract.Data` 行时“向后引用”该值。
>
> 系统会在您首次调用 `applyBatch()` 时创建整个结果数组，其大小与您提供的 `ContentProviderOperation` 对象的 `ArrayList` 大小相等。 不过，结果数组中的所有元素都设置为 `null`，如果您试图向后引用某个尚未应用的操作的结果， `withValueBackReference()` 会引发一个 `Exception`。

以下代码段说明如何批量插入新原始联系人和数据。代码段中包括用于建立屈服点和使用向后引用的代码。 这些代码段是扩展版本的 `createContacEntry()` 方法，该方法是 `Contact Manager` 示例应用中 `ContactAdder` 类的组成部分。

第一个代码段用于检索 UI 中的联系人数据。此时，用户已经选择了应添加新原始联系人的帐户。

```
// Creates a contact entry from the current UI values, using the currently-selected account.
protected void createContactEntry() {
    /*
     * Gets values from the UI
     */
    String name = mContactNameEditText.getText().toString();
    String phone = mContactPhoneEditText.getText().toString();
    String email = mContactEmailEditText.getText().toString();

    int phoneType = mContactPhoneTypes.get(
            mContactPhoneTypeSpinner.getSelectedItemPosition());

    int emailType = mContactEmailTypes.get(
            mContactEmailTypeSpinner.getSelectedItemPosition());
```

下一个代码段用于创建将该原始联系人行插入 `ContactsContract.RawContacts` 表的操作：

```
    /*
     * Prepares the batch operation for inserting a new raw contact and its data. Even if
     * the Contacts Provider does not have any data for this person, you can't add a Contact,
     * only a raw contact. The Contacts Provider will then add a Contact automatically.
     */

     // Creates a new array of ContentProviderOperation objects.
    ArrayList<ContentProviderOperation> ops =
            new ArrayList<ContentProviderOperation>();

    /*
     * Creates a new raw contact with its account type (server type) and account name
     * (user's account). Remember that the display name is not stored in this row, but in a
     * StructuredName data row. No other data is required.
     */
    ContentProviderOperation.Builder op =
            ContentProviderOperation.newInsert(ContactsContract.RawContacts.CONTENT_URI)
            .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, mSelectedAccount.getType())
            .withValue(ContactsContract.RawContacts.ACCOUNT_NAME, mSelectedAccount.getName());

    // Builds the operation and adds it to the array of operations
    ops.add(op.build());
```

接着，代码会创建显示姓名行、电话行和电子邮件行的数据行。

每个操作生成器对象都使用 `withValueBackReference()` 来获取 `RAW_CONTACT_ID`。引用指回来自第一次操作的 `ContentProviderResult` 对象，第一次操作就是添加原始联系人行并返回其新 `_ID` 值。 结果是，每个数据行都通过其 `RAW_CONTACT_ID` 自动链接到其所属的 `ContactsContract.RawContacts` 行。

添加电子邮件行的 `ContentProviderOperation.Builder` 对象带有 `withYieldAllowed()` 标志，用于设置屈服点：

```
    // Creates the display name for the new raw contact, as a StructuredName data row.
    op =
            ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
            /*
             * withValueBackReference sets the value of the first argument to the value of
             * the ContentProviderResult indexed by the second argument. In this particular
             * call, the raw contact ID column of the StructuredName data row is set to the
             * value of the result returned by the first operation, which is the one that
             * actually adds the raw contact row.
             */
            .withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)

            // Sets the data row's MIME type to StructuredName
            .withValue(ContactsContract.Data.MIMETYPE,
                    ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE)

            // Sets the data row's display name to the name in the UI.
            .withValue(ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME, name);

    // Builds the operation and adds it to the array of operations
    ops.add(op.build());

    // Inserts the specified phone number and type as a Phone data row
    op =
            ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
            /*
             * Sets the value of the raw contact id column to the new raw contact ID returned
             * by the first operation in the batch.
             */
            .withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)

            // Sets the data row's MIME type to Phone
            .withValue(ContactsContract.Data.MIMETYPE,
                    ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)

            // Sets the phone number and type
            .withValue(ContactsContract.CommonDataKinds.Phone.NUMBER, phone)
            .withValue(ContactsContract.CommonDataKinds.Phone.TYPE, phoneType);

    // Builds the operation and adds it to the array of operations
    ops.add(op.build());

    // Inserts the specified email and type as a Phone data row
    op =
            ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
            /*
             * Sets the value of the raw contact id column to the new raw contact ID returned
             * by the first operation in the batch.
             */
            .withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)

            // Sets the data row's MIME type to Email
            .withValue(ContactsContract.Data.MIMETYPE,
                    ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE)

            // Sets the email address and type
            .withValue(ContactsContract.CommonDataKinds.Email.ADDRESS, email)
            .withValue(ContactsContract.CommonDataKinds.Email.TYPE, emailType);

    /*
     * Demonstrates a yield point. At the end of this insert, the batch operation's thread
     * will yield priority to other threads. Use after every set of operations that affect a
     * single contact, to avoid degrading performance.
     */
    op.withYieldAllowed(true);

    // Builds the operation and adds it to the array of operations
    ops.add(op.build());
```

最后一个代码段显示的是 `applyBatch()` 调用，用于插入新原始联系人行和数据行。

```
    // Ask the Contacts Provider to create a new contact
    Log.d(TAG,"Selected account: " + mSelectedAccount.getName() + " (" +
            mSelectedAccount.getType() + ")");
    Log.d(TAG,"Creating contact: " + name);

    /*
     * Applies the array of ContentProviderOperation objects in batch. The results are
     * discarded.
     */
    try {

            getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);
    } catch (Exception e) {

            // Display a warning
            Context ctx = getApplicationContext();

            CharSequence txt = getString(R.string.contactCreationFailure);
            int duration = Toast.LENGTH_SHORT;
            Toast toast = Toast.makeText(ctx, txt, duration);
            toast.show();

            // Log exception
            Log.e(TAG, "Exception encountered while inserting contact: " + e);
    }
}
```

此外，您还可以利用批处理操作实现**乐观并发控制**，这是一种无需锁定底层存储区便可应用修改事务的控制方法。 要使用此方法，您需要应用事务，然后检查是否存在可能已同时做出的其他修改。 如果您发现了不一致的修改，请回滚事务并重试。

乐观并发控制对于移动设备很有用，因为在移动设备上，同一时间只有一位用户，并且同时访问数据存储区的情况很少见。 由于未使用锁定功能，因此不用浪费时间设置锁定或等待其他事务解除锁定。

要在更新某个 `ContactsContract.RawContacts` 行时使用乐观并发控制，请按以下步骤操作：

1. 检索原始联系人的 `VERSION` 列以及要检索的其他数据。
2. 创建一个适合使用 `newAssertQuery(Uri)` 方法强制执行约束 的 `ContentProviderOperation.Builder` 对象。对于内容 URI，请使用追加有原始联系人 `_ID` 的 `RawContacts.CONTENT_URI` 。
3. 对于 `ContentProviderOperation.Builder` 对象，请调用 `withValue()`，对 `VERSION` 列与您刚检索的版本号进行比较。
4. 对于同一 `ContentProviderOperation.Builder`，请调用 `withExpectedCount()`，确保此断言只对一行进行测试。
5. 调用 `build()` 创建 `ContentProviderOperation` 对象，然后将此对象添加为要传递至 `applyBatch()` 的 `ArrayList` 中的第一个对象。
6. 应用批处理事务。

如果在您读取原始联系人行到您试图对其进行修改这段时间有另一项操作更新了该行，“断言”`ContentProviderOperation` 将会失败，系统将终止整个批处理操作。 此情况下，您可以选择重新执行批处理操作，或执行其他某操作。

以下代码段演示如何在使用 `CursorLoader` 查询一位原始联系人后创建一个“断言” `ContentProviderOperation`：

```
/*
 * The application uses CursorLoader to query the raw contacts table. The system calls this method
 * when the load is finished.
 */
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {

    // Gets the raw contact's _ID and VERSION values
    mRawContactID = cursor.getLong(cursor.getColumnIndex(BaseColumns._ID));
    mVersion = cursor.getInt(cursor.getColumnIndex(SyncColumns.VERSION));
}

...

// Sets up a Uri for the assert operation
Uri rawContactUri = ContentUris.withAppendedId(RawContacts.CONTENT_URI, mRawContactID);

// Creates a builder for the assert operation
ContentProviderOperation.Builder assertOp = ContentProviderOperation.netAssertQuery(rawContactUri);

// Adds the assertions to the assert operation: checks the version and count of rows tested
assertOp.withValue(SyncColumns.VERSION, mVersion);
assertOp.withExpectedCount(1);

// Creates an ArrayList to hold the ContentProviderOperation objects
ArrayList ops = new ArrayList<ContentProviderOperationg>;

ops.add(assertOp.build());

// You would add the rest of your batch operations to "ops" here

...

// Applies the batch. If the assert fails, an Exception is thrown
try
    {
        ContentProviderResult[] results =
                getContentResolver().applyBatch(AUTHORITY, ops);

    } catch (OperationApplicationException e) {

        // Actions you want to take if the assert operation fails go here
    }
```

通过 Intent 执行检索和修改

通过向设备的联系人应用发送 Intent，您可以间接访问联系人提供程序。 Intent 会启动设备的联系人应用 UI，用户可以在其中执行与联系人有关的操作。通过这种访问方式，用户可以：

- 从列表中选取一位联系人并将其返回给您的应用以执行进一步操作。
- 编辑现有联系人的数据。
- 为其任一帐户插入新原始联系人。
- 删除联系人或联系人数据。

如果用户要插入或更新数据，您可以先收集数据，然后将其作为 Intent 的一部分发送。

当您使用 Intent 通过设备的联系人应用访问联系人提供程序时，您无需自行编写用于访问该提供程序的 UI 或代码。 您也无需请求对提供程序的读取或写入权限。 设备的联系人应用可以将联系人读取权限授予给您，而且您是通过另一个应用对该提供程序进行修改，不需要拥有写入权限。

内容提供程序基础知识指南的“通过 Intent 访问数据”部分详细描述了通过发送 Intent 来访问某提供程序的一般过程。 表 4 汇总了您为可用任务使用的操作、MIME 类型以及数据值，`ContactsContract.Intents.Insert` 参考文档列出了您可用于`putExtra()` 的 Extra 值：

表 4. 联系人提供程序 Intent。

任务	操作	数据	MIME 类型	说明
从列表中选取一位联系人	`ACTION_PICK`	下列值之一： • `Contacts.CONTENT_URI`，显示联系人列表。 • `Phone.CONTENT_URI`，显示原始联系人的电话号码列表。	未使用	显示原始联系人列表或一位原始联系人的数据列表，具体取决于您提供的内容 URI 类型。调用 `startActivityForResult()` 方法，该方法返回所选行的内

		• `StructuredPostal.CONTENT_URI`，显示原始联系人的邮政地址列表。 • `Email.CONTENT_URI`，显示原始联系人的电子邮件地址列表。		容 URI。 该 URI 的形式为：追加有该行 `LOOKUP_ID` 的表的内容 URI。 设备的联系人应用会在 Activity 的生命周期内将读取和写入权限授予给此内容 URI。 如需了解更多详细信息，请参阅内容提供程序基础知识指南。
插入新原始联系人	`Insert.ACTION`	不适用	`RawContacts.CONTENT_TYPE`，用于一组原始联系人的 MIME 类型。	显示设备“通讯录”应用的 **Add Contact** 屏幕。系统会显示您添加到 Intent 中的 Extra 值。如果是随 `startActivityForResult()` 发送，系统会将新添加的原始联系人的内容 URI 传回给 Activity 的 `onActivityResult()` 回调方法并作为后者 `Intent` 参数的“data”字段。 要获取该值，请调用 `getData()`。
编辑联系人	`ACTION_EDIT`	该联系人的 `CONTENT_LOOKUP_URI`。 该编辑器 Activity 让用户能够对任何与该联系人关联的数据进行编辑。	`Contacts.CONTENT_ITEM_TYPE`，一位联系人。	显示“通讯录”应用中的“Edit Contact”屏幕。系统会显示您添加到 Intent 中的 Extra 值。当用户点击 **Done** 保存编辑时，您的 Activity 会返回前台。
显示一个同样可以添加数据的选取器。	`ACTION_INSERT_OR_EDIT`	不适用	`CONTENT_ITEM_TYPE`	此 Intent 始终显示“通讯录”应用的选取器屏幕。用户可以选取要编辑的联系人，或添加新联系人。 根据用户的选择，系统会显示编辑屏幕或添加屏幕，还会显示您使用 Intent 传递的 Extra 数据。 如果您的应用显示电子邮件或电话号码等联系人数据，请使用此 Intent 来允许用户向现有联系人添加数据。 **注：**不需要通过此 Intent 的 Extra 发送姓名值，因为用户总是会选取现有姓名或添加新姓名。 此外，如果您发送姓名，并且用户选择执行编辑操作，则联系人应用将显示您发送的姓名，该姓名将覆盖以前的值。 如果用户未注意这一情况便保存了编辑，原有值将会丢失。

设备的联系人应用不允许您使用 Intent 删除原始联系人或其任何数据。 因此，要删除原始联系人，请使用 `ContentResolver.delete()` 或 `ContentProviderOperation.newDelete()`。

以下代码段说明如何构建和发送一个插入新原始联系人和数据的 Intent：

```
// Gets values from the UI
String name = mContactNameEditText.getText().toString();
String phone = mContactPhoneEditText.getText().toString();
String email = mContactEmailEditText.getText().toString();
```

```
String company = mCompanyName.getText().toString();
String jobtitle = mJobTitle.getText().toString();

// Creates a new intent for sending to the device's contacts application
Intent insertIntent = new Intent(ContactsContract.Intents.Insert.ACTION);

// Sets the MIME type to the one expected by the insertion activity
insertIntent.setType(ContactsContract.RawContacts.CONTENT_TYPE);

// Sets the new contact name
insertIntent.putExtra(ContactsContract.Intents.Insert.NAME, name);

// Sets the new company and job title
insertIntent.putExtra(ContactsContract.Intents.Insert.COMPANY, company);
insertIntent.putExtra(ContactsContract.Intents.Insert.JOB_TITLE, jobtitle);

/*
 * Demonstrates adding data rows as an array list associated with the DATA key
 */

// Defines an array list to contain the ContentValues objects for each row
ArrayList<ContentValues> contactData = new ArrayList<ContentValues>();


/*
 * Defines the raw contact row
 */

// Sets up the row as a ContentValues object
ContentValues rawContactRow = new ContentValues();

// Adds the account type and name to the row
rawContactRow.put(ContactsContract.RawContacts.ACCOUNT_TYPE, mSelectedAccount.getType());
rawContactRow.put(ContactsContract.RawContacts.ACCOUNT_NAME, mSelectedAccount.getName());

// Adds the row to the array
contactData.add(rawContactRow);

/*
 * Sets up the phone number data row
 */

// Sets up the row as a ContentValues object
ContentValues phoneRow = new ContentValues();

// Specifies the MIME type for this data row (all data rows must be marked by their type)
phoneRow.put(
        ContactsContract.Data.MIMETYPE,
        ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE
);

// Adds the phone number and its type to the row
phoneRow.put(ContactsContract.CommonDataKinds.Phone.NUMBER, phone);

// Adds the row to the array
contactData.add(phoneRow);

/*
 * Sets up the email data row
 */

// Sets up the row as a ContentValues object
ContentValues emailRow = new ContentValues();

// Specifies the MIME type for this data row (all data rows must be marked by their type)
emailRow.put(
        ContactsContract.Data.MIMETYPE,
        ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE
);

// Adds the email address and its type to the row
emailRow.put(ContactsContract.CommonDataKinds.Email.ADDRESS, email);

// Adds the row to the array
contactData.add(emailRow);

/*
 * Adds the array to the intent's extras. It must be a parcelable object in order to
 * travel between processes. The device's contacts app expects its key to be
 * Intents.Insert.DATA
```

```
 */
insertIntent.putParcelableArrayListExtra(ContactsContract.Intents.Insert.DATA, contactData);

// Send out the intent to start the device's contacts app in its add contact activity.
startActivity(insertIntent);
```

数据完整性

联系人存储区包含用户认为是正确且是最新的重要敏感数据，因此联系人提供程序具有规定清晰的数据完整性规则。 您有责任在修改联系人数据时遵守这些规则。 以下列出了其中的重要规则：

务必为您添加的每个 `ContactsContract.RawContacts` 行添加一个 `ContactsContract.CommonDataKinds.StructuredName` 行。

> 如果 `ContactsContract.Data` 表中的 `ContactsContract.RawContacts` 行没有 `ContactsContract.CommonDataKinds.StructuredName` 行，可能会在聚合时引发问题。

务必将新 `ContactsContract.Data` 行链接到其父 `ContactsContract.RawContacts` 行。

> 如果 `ContactsContract.Data` 行未链接到 `ContactsContract.RawContacts`，则其在设备的联系人应用中将处于不可见状态，而且这可能会导致同步适配器出现问题。

请仅更改您拥有的那些原始联系人的数据。

> 请切记，联系人提供程序所管理的数据通常来自多个不同帐户类型/在线服务。 您需要确保您的应用仅修改或删除归您所有的行的数据，并且仅通过您控制的帐户类型和帐户名称插入数据。

务必使用在 `ContactsContract` 及其子类中为权限、内容 URI、URI 路径、列名称、MIME 类型以及 `TYPE` 值定义的常量。

> 使用这些常量有助于您避免错误。如有任何常量被弃用，您还会从编译器警告收到通知。

自定义数据行

通过创建和使用自己的自定义 MIME 类型，您可以在 `ContactsContract.Data` 表中插入、编辑、删除和检索您的自有数据行。 这些行仅限使用 `ContactsContract.DataColumns` 中定义的列，但您可以将您自己的类型专用列名称映射到默认列名称。 在设备的联系人应用中，会显示这些行的数据，但无法对其进行编辑或删除，用户也无法添加其他数据。 要允许用户修改您的自定义数据行，您必须在自己的应用中提供编辑器 Activity。

要显示您的自定义数据，请提供一个 `contacts.xml` 文件，其中须包含一个 `<ContactsAccountType>` 元素，及其一个或多个 `<ContactsDataKind>` 子元素。`<ContactsDataKind> element` 部分对此做了更详尽的描述。

如需了解有关自定义 MIME 类型的更多信息，请阅读创建内容提供程序指南。

联系人提供程序同步适配器

联系人提供程序专门设计用于处理设备与在线服务之间的联系人数据**同步**。 借助同步功能，用户可以将现有数据下载到新设备，以及将现有数据上传到新帐户。 此外，同步还能确保用户掌握最新数据，无需考虑数据增加和更改的来源。 同步的另一个优点是，即使设备未连接网络，联系人数据同样可用。

虽然您可以通过各种方式实现同步，不过 Android 系统提供了一个插件同步框架，可自动化完成下列任务：

- 检查网络可用性。
- 根据用户偏好安排和执行同步。
- 重启已停止的同步。

要使用此框架，您需要提供一个同步适配器插件。每个同步适配器都专用于某个服务和内容提供程序，但可以处理同一服务的多个帐户名称。 该框架还允许同一服务和提供程序具有多个同步适配器。

同步适配器类和文件

您需要将同步适配器作为 `AbstractThreadedSyncAdapter` 的子类进行实现，并作为 Android 应用的一部分进行安装。系统通过您的应用清单文件中的元素以及由清单文件指向的一个特殊 XML 文件了解有关同步适配器的信息。 该 XML 文件定义在线服务的帐户类型和内容提供程序的权限，它们共同对适配器进行唯一标识。 用户为同步适配器的帐户类型添加一个帐户，并为与同步适配器同步的内容提供程序启用同步后，同步适配器才会激活。 激活后，系统将开始管理适配器，并在必要时调用它，以在内容提供程序与服务器之间同步数据。

> **注：**将帐户类型用作同步适配器标识的一部分让系统可以发现从同一组织访问不同服务的同步适配器，并将它们组合在一起。 例如，Google 在线服

> 务的同步适配器都具有相同的帐户类型 `com.google`。 当用户向其设备添加 Google 帐户时，已安装的所有 Google 服务同步适配器将一起列出；列出的每个同步适配器都与设备上不同的内容提供程序同步。

大多数服务都要求用户验证身份后才能访问数据，为此，Android 系统提供了一个身份验证框架，该框架与同步适配器框架类似，并且经常与其联用。 该身份验证框架使用的插件身份验证器是 `AbstractAccountAuthenticator` 的子类。 身份验证器通过下列步骤验证用户的身份：

1. 收集用户名、用户密码或类似信息（用户的**凭据**）。
2. 将凭据发送给服务
3. 检查服务的回复。

如果服务接受了凭据，身份验证器便可存储凭据以供日后使用。 由于插件身份验证器框架的存在，`AccountManager` 可以提供对身份验证器支持并选择公开的任何身份验证令牌（例如 OAuth2 身份验证令牌）的访问。

尽管身份验证并非必需，但大多数联系人服务都会使用它。 不过，您不一定要使用 Android 身份验证框架进行身份验证。

同步适配器实现

要为联系人提供程序实现同步适配器，您首先要创建一个包含以下内容的 Android 应用：

一个 `Service` 组件，用于响应系统发出的绑定到同步适配器的请求。

> 当系统想要运行同步时，它会调用服务的 `onBind()` 方法，为同步适配器获取一个 `IBinder`。这样，系统便可跨进程调用适配器的方法。
>
> 在示例同步适配器示例应用中，该服务的类名是 `com.example.android.samplesync.syncadapter.SyncService`。

作为 `AbstractThreadedSyncAdapter` 具体子类实现的实际同步适配器。

> 此类的作用是从服务器下载数据、从设备上传数据以及解决冲突。 适配器的主要工作是在方法 `onPerformSync()` 中完成的。 必须将此类实例化为单一实例。
>
> 在示例同步适配器示例应用中，同步适配器是在 `com.example.android.samplesync.syncadapter.SyncAdapter` 类中定义的。

`Application` 的子类。

> 此类充当同步适配器单一实例的工厂。使用 `onCreate()` 方法实例化同步适配器，并提供一个静态“getter”方法，使单一实例返回同步适配器服务的 `onBind()` 方法。

可选：一个 `Service` 组件，用于响应系统发出的用户身份验证请求。

> `AccountManager` 会启动此服务以开始身份验证流程。 该服务的 `onCreate()` 方法会将一个身份验证器对象实例化。 当系统想要对应用同步适配器的用户帐户进行身份验证时，它会调用该服务的 `onBind()` 方法，为该身份验证器获取一个 `IBinder`。 这样，系统便可跨进程调用身份验证器的方法。
>
> 在示例同步适配器示例应用中，该服务的类名是 `com.example.android.samplesync.authenticator.AuthenticationService`。

可选：一个用于处理身份验证请求的 `AbstractAccountAuthenticator` 具体子类。

> `AccountManager` 就是调用此类所提供的方法向服务器验证用户的凭据。 详细的身份验证过程会因服务器所采用技术的不同而有很大差异。 您应该参阅服务器软件的文档，了解有关身份验证的更多信息。
>
> 在示例同步适配器示例应用中，身份验证器是在 `com.example.android.samplesync.authenticator.Authenticator` 类中定义的。

用于定义系统同步适配器和身份验证器的 XML 文件。

> 之前描述的同步适配器和身份验证器服务组件都是在应用清单文件中的 `<service>` 元素内定义的。 这些元素包含以下用于向系统提供特定数据的 `<meta-data>` 子元素：
>
> - 同步适配器服务的 `<meta-data>` 元素指向 XML 文件 `res/xml/syncadapter.xml`。而该文件则指定将与联系人提供程序同步的网络服务的 URI，以及指定该 Web 服务的帐户类型。
> - **可选：**身份验证器的 `<meta-data>` 元素指向 XML 文件 `res/xml/authenticator.xml`。而该文件则指定此身份验证器所支持的帐户类型，以及指定身份验证过程中出现的 UI 资源。 在此元素中指定的帐户类型必须与为同步适配器指定的帐户类型相同。

社交流数据

android.provider.ContactsContract.StreamItems 表和 android.provider.ContactsContract.StreamItemPhotos 表管理来自社交网络的传入数据。 您可以编写一个同步适配器，用其将您自己社交网络中的流数据添加到这些表中，也可以从这些表读取流数据并将其显示在您的自有应用中，或者同时采用这两种

方法。 利用这些功能，可以将您的社交网络服务和应用集成到 Android 的社交网络体验之中。

社交流文本

流项目始终与原始联系人关联。android.provider.ContactsContract.StreamItemsColumns#RAW_CONTACT_ID 链接到原始联系人的 `_ID` 值。 原始联系人的帐户类型和帐户名称也存储在流项目行中。

将您的流数据存储在以下列中：

android.provider.ContactsContract.StreamItemsColumns#ACCOUNT_TYPE

> **必备。**与该流项目关联的原始联系人对应的用户帐户类型。 请记得在插入流项目时设置此值。

android.provider.ContactsContract.StreamItemsColumns#ACCOUNT_NAME

> **必备。**与该流项目关联的原始联系人对应的用户帐户名称。 请记得在插入流项目时设置此值。

标识符列

> **必备。**您必须在插入流项目时插入下列标识符列：
>
> - android.provider.ContactsContract.StreamItemsColumns#CONTACT_ID：此流项目关联的联系人的 android.provider.BaseColumns#_ID 值。
> - android.provider.ContactsContract.StreamItemsColumns#CONTACT_LOOKUP_KEY：此流项目关联的联系人的 android.provider.ContactsContract.ContactsColumns#LOOKUP_KEY 值。
> - android.provider.ContactsContract.StreamItemsColumns#RAW_CONTACT_ID：此流项目关联的原始联系人的 android.provider.BaseColumns#_ID 值。

android.provider.ContactsContract.StreamItemsColumns#COMMENTS

> 可选。存储可在流项目开头显示的摘要信息。

android.provider.ContactsContract.StreamItemsColumns#TEXT

> 流项目的文本，或为项目来源发布的内容，或是对生成流项目的某项操作的描述。 此列可包含可由 `fromHtml()` 渲染的任何格式设置和嵌入式资源图像。 提供程序可能会截断或省略较长内容，但它会尽力避免破坏标记。

android.provider.ContactsContract.StreamItemsColumns#TIMESTAMP

> 一个包含流项目插入时间或更新时间的文本字符串，以从公元纪年开始计算的*毫秒数*形式表示。 此列由插入或更新流项目的应用负责维护；联系人提供程序不会自动对其进行维护。

要显示您的流项目的标识信息，请使用 android.provider.ContactsContract.StreamItemsColumns#RES_ICON、android.provider.ContactsContract.StreamItemsColumns#RES_LABEL 和 android.provider.ContactsContract.StreamItemsColumns#RES_PACKAGE 链接到您的应用中的资源。

android.provider.ContactsContract.StreamItems 表还包含供同步适配器专用的列 android.provider.ContactsContract.StreamItemsColumns#SYNC1 至 android.provider.ContactsContract.StreamItemsColumns#SYNC4。

社交流照片

android.provider.ContactsContract.StreamItemPhotos 表存储与流项目关联的照片。 该表的 android.provider.ContactsContract.StreamItemPhotosColumns#STREAM_ITEM_ID 列链接到 android.provider.ContactsContract.StreamItems 表的 `_ID` 列中的值。 照片引用存储在表中的以下列：

android.provider.ContactsContract.StreamItemPhotos#PHOTO 列（一个二进制大型对象）。

> 照片的二进制表示，为便于存储和显示，由提供程序调整了尺寸。 此列可用于向后兼容使用它来存储照片的旧版本联系人提供程序。 不过，在当前版本中，您不应使用此列来存储照片， 而应使用 android.provider.ContactsContract.StreamItemPhotosColumns#PHOTO_FILE_ID 或 android.provider.ContactsContract.StreamItemPhotosColumns#PHOTO_URI（下文对两者都做了描述）将照片存储在一个文件内。 此列现在包含可用于读取的照片缩略图。

android.provider.ContactsContract.StreamItemPhotosColumns#PHOTO_FILE_ID

> 原始联系人照片的数字标识符。将此值追加到常量 `DisplayPhoto.CONTENT_URI`，获取指向单一照片文件的内容 URI，然后调用 `openAssetFileDescriptor()` 来获取照片文件的句柄。

android.provider.ContactsContract.StreamItemPhotosColumns#PHOTO_URI

一个内容 URI，直接指向此行所表示的照片的照片文件。 通过此 URI 调用 `openAssetFileDescriptor()` 以获得照片文件的句柄。

使用社交流表

这些表的工作方式与联系人提供程序中的其他主表基本相同，不同的是：

- 这些表需要额外的访问权限。要读取它们的数据，您的应用必须具有 android.Manifest.permission#READ_SOCIAL_STREAM 权限。 要修改它们，您的应用必须具有 android.Manifest.permission#WRITE_SOCIAL_STREAM 权限。
- 对于 android.provider.ContactsContract.StreamItems 表，为每一位原始联系人存储的行数有限。 一旦达到该限制，联系人提供程序即会自动删除 android.provider.ContactsContract.StreamItemsColumns#TIMESTAMP 最早的行，为新流项目行腾出空间。 要获取该限制，请发出对内容 URI android.provider.ContactsContract.StreamItems#CONTENT_LIMIT_URI 的查询。 您可以将内容 URI 以外的所有其他参数保持设置为 `null`。 查询会返回一个 Cursor，其中包含一行，并且只有 android.provider.ContactsContract.StreamItems#MAX_ITEMS 一列。

android.provider.ContactsContract.StreamItems.StreamItemPhotos 类定义了 android.provider.ContactsContract.StreamItemPhotos 的一个子表，其中包含某个流项目的照片行。

社交流交互

通过将联系人提供程序管理的社交流数据与设备的联系人应用相结合，可以在您的社交网络系统与现有联系人之间建立起有效的连接。 这种结合实现了下列功能：

- 您可以通过同步适配器让您的社交网络服务与联系人提供程序同步，检索用户联系人的近期 Activity，并将其存储在 android.provider.ContactsContract.StreamItems 表和 android.provider.ContactsContract.StreamItemPhotos 表中，以供日后使用。
- 除了定期同步外，您还可以在用户选择某位联系人进行查看时触发您的同步适配器以检索更多数据。 这样，您的同步适配器便可检索该联系人的高分辨率照片和最近流项目。
- 通过在设备的联系人应用以及联系人提供程序中注册通知功能，您可以在用户查看联系人时*收到*一个 Intent，并在那时通过您的服务更新联系人的状态。 与通过同步适配器执行完全同步相比，此方法可能更快速，占用的带宽也更少。
- 用户可以在查看设备联系人应用中的联系人时，将其添加到您的社交网络服务。 您可以通过“邀请联系人”功能实现此目的，而该功能则是通过将 Activity 与 XML 文件结合使用来实现的，前者将现有联系人添加到您的社交网络，后者为设备的联系人应用以及联系人提供程序提供有关您的应用的详细信息。

流项目与联系人提供程序的定期同步与其他同步相同。 如需了解有关同步的更多信息，请参阅 联系人提供程序同步适配器部分。接下来的两节介绍如何注册通知和邀请联系人。

通过注册处理社交网络查看

要注册您的同步适配器，以便在用户查看由您的同步适配器管理的联系人时收到通知，请执行以下步骤：

1. 在您项目的 `res/xml/` 目录中创建一个名为 `contacts.xml` 的文件。 如果您已有该文件，可跳过此步骤。
2. 在该文件中添加元素 `<ContactsAccountType xmlns:android="http://schemas.android.com/apk/res/android">`。 如果该元素已存在，可跳过此步骤。
3. 要注册一项服务，以便在用户于设备的联系人应用中打开某位联系人的详细信息页面时通知该服务，请为该元素添加 `viewContactNotifyService="`*`serviceclass`*`"` 属性，其中 *`serviceclass`* 是该服务的完全限定类名，应由该服务接收来自设备联系人应用的 Intent。 对于这个通知程序服务，请使用一个扩展 `IntentService` 的类，以让该服务能够接收 Intent。 传入 Intent 中的数据包含用户点击的原始联系人的内容 URI。 您可以通过通知程序服务绑定到您的同步适配器，然后调用同步适配器来更新原始联系人的数据。

要注册需要在用户点击流项目或照片（或同时点击这两者）时调用的 Activity，请执行以下步骤：

1. 在您项目的 `res/xml/` 目录中创建一个名为 `contacts.xml` 的文件。 如果您已有该文件，可跳过此步骤。
2. 在该文件中添加元素 `<ContactsAccountType xmlns:android="http://schemas.android.com/apk/res/android">`。 如果该元素已存在，可跳过此步骤。
3. 要注册某个 Activity，以处理用户在设备联系人应用中点击某个流项目的操作，请为该元素添加 `viewStreamItemActivity="`*`activityclass`*`"` 属性，其中 *`activityclass`* 是该 Activity 的完全限定类名，应由该 Activity 接收来自设备联系人应用的 Intent。
4. 要注册某个 Activity，以处理用户在设备联系人应用中点击某个流照片的操作，请为该元素添加 `viewStreamItemPhotoActivity="`*`activityclass`*`"` 属性，其中 *`activityclass`* 是该 Activity 的完全限定类名，应由该 Activity 接收来自设备联系人应用的 Intent。

<ContactsAccountType> 元素部分对 `<ContactsAccountType>` 元素做了更详尽的描述。

传入 Intent 包含用户点击的项目或照片的内容 URI。 要让文本项目和照片具有独立的 Activity，请在同一文件中使用这两个属性。

与您的社交网络服务交互

用户不必为了邀请联系人到您的社交网络网站而离开设备的联系人应用。 取而代之是，您可以让设备的联系人应用发送一个 Intent，将联系人 邀请到您的 Activity 之一。要设置此功能，请执行以下步骤：

1. 在您项目的 `res/xml/` 目录中创建一个名为 `contacts.xml` 的文件。 如果您已有该文件，可跳过此步骤。
2. 在该文件中添加元素 `<ContactsAccountType xmlns:android="http://schemas.android.com/apk/res/android">`。 如果该元素已存在，可跳过此步骤。
3. 添加以下属性：
   - `inviteContactActivity="activityclass"`
   - `inviteContactActionLabel="@string/invite_action_label"`

   `activityclass` 值是应该接收该 Intent 的 Activity 的完全限定类名。 *`invite_action_label`* 值是一个文本字符串，将显示在设备联系人应用的 **Add Connection** 菜单中。

> **注：**`ContactsSource` 是 `ContactsAccountType` 的一个已弃用的标记名称。

contacts.xml 引用

文件 `contacts.xml` 包含一些 XML 元素，这些元素控制您的同步适配器和应用与联系人应用及联系人提供程序的交互。 下文对这些元素做了描述。

<ContactsAccountType> 元素

`<ContactsAccountType>` 元素控制您的应用与联系人应用的交互。 它采用了以下语法：

```
<ContactsAccountType
        xmlns:android="http://schemas.android.com/apk/res/android"
        inviteContactActivity="activity_name"
        inviteContactActionLabel="invite_command_text"
        viewContactNotifyService="view_notify_service"
        viewGroupActivity="group_view_activity"
        viewGroupActionLabel="group_action_text"
        viewStreamItemActivity="viewstream_activity_name"
        viewStreamItemPhotoActivity="viewphotostream_activity_name">
```

包含它的文件：

`res/xml/contacts.xml`

可包含：

`<ContactsDataKind>`

说明：

声明 Android 组件和 UI 标签，让用户能够邀请他们的一位联系人加入社交网络，在他们的某个社交网络流更新时通知用户，以及执行其他操作。

请注意，对 `<ContactsAccountType>` 的属性而言，属性前缀 `android:` 并非必需的。

属性：

`inviteContactActivity`

> 您的应用中某个 Activity 的完全限定类名，您想要在用户于设备的联系人应用中选择 **Add connection** 时激活该 Activity。

`inviteContactActionLabel`

> **Add connection** 菜单中为 `inviteContactActivity` 中指定的 Activity 显示的文本字符串。 例如，您可以使用字符串“Follow in my network”。您可以为此标签使用字符串资源标识符。

`viewContactNotifyService`

> 您的应用中某项服务的完全限定类名，当用户查看联系人时，应由该服务接收通知。 此通知由设备的联系人应用发送；您的应用可以根据通知将数据密集型操作推迟到必要时再执行。 例如，您的应用对此通知的响应可以是：读入并显示联系人的高分辨率照片和最近的社交流项目。 社交流交互部分对此功能做了更详尽的描述。 您可以在 SampleSyncAdapter 示例应用的 `NotifierService.java` 文件中查看通知服务的示例。

`viewGroupActivity`

您的应用中某个可显示组信息的 Activity 的完全限定类名。 当用户点击设备联系人应用中的组标签时，将显示此 Activity 的 UI。

`viewGroupActionLabel`

联系人应用为某个 UI 控件显示的标签，用户可通过该控件查看您的应用中的组。

例如，如果您在设备上安装了 Google+ 应用，并将 Google+ 与联系人应用同步，就会看到 Google+ 圈子以组的形式出现在您的联系人应用的 **Groups** 选项卡内。 如果您点击某个 Google+ 圈子，就会看到该圈子内的联系人以“组”的形式列出。 在该显示页面的顶部，您会看到一个 Google+ 图标；如果您点击它，控制权将切换给 Google+ 应用。 “通讯录”应用以 Google+ 图标作为 `viewGroupActionLabel` 的值，通过 `viewGroupActivity` 来实现此目的。

允许使用字符串资源标识符作为该属性的值。

`viewStreamItemActivity`

您的应用中某个 Activity 的完全限定类名，设备的联系人应用会在用户点击原始联系人的流项目时启动该 Activity。

`viewStreamItemPhotoActivity`

您的应用中某个 Activity 的完全限定类名，设备的联系人应用会在用户点击原始联系人流项目中的照片时启动该 Activity。

<ContactsDataKind> 元素

`<ContactsDataKind>` 元素控制您的应用的自定义数据行在联系人应用 UI 中的显示。 它采用了以下语法：

```
<ContactsDataKind
        android:mimeType="MIMEtype"
        android:icon="icon_resources"
        android:summaryColumn="column_name"
        android:detailColumn="column_name">
```

包含它的文件：

`<ContactsAccountType>`

说明：

此元素用于让联系人应用将自定义数据行的内容显示为原始联系人详细信息的一部分。 `<ContactsAccountType>` 的每个 `<ContactsDataKind>` 子元素都代表您的同步适配器向 `ContactsContract.Data` 表添加的某个自定义数据行类型。 请为您使用的每个自定义 MIME 类型添加一个 `<ContactsDataKind>` 元素。 如果您不想显示任何自定义数据行的数据，则无需添加该元素。

属性：

`android:mimeType`

您为 `ContactsContract.Data` 表中某个自定义数据行类型定义的自定义 MIME 类型。例如，可将值 `vnd.android.cursor.item/vnd.example.locationstatus` 作为记录联系人最后已知位置的数据行的自定义 MIME 类型。

`android:icon`

联系人应用在您的数据旁显示的 Android Drawable资源。 它用于向用户指示数据来自您的服务。

`android:summaryColumn`

从数据行检索的两个值中第一个值的列名。该值显示为该数据行的第一个输入行。 第一行专用作数据摘要，不过它是可选项。 另请参阅 android:detailColumn。

`android:detailColumn`

从数据行检索的两个值中第二个值的列名。该值显示为该数据行的第二个输入行。 另请参阅 `android:summaryColumn`。

其他联系人提供程序功能

除了上文描述的主要功能外，联系人提供程序还为处理联系人数据提供了下列有用的功能：

- 联系人组
- 照片功能

联系人组

联系人组

联系人提供程序可以选择性地为相关联系人集合添加**组**数据标签。 如果与某个用户帐户关联的服务器想要维护组，则与该帐户的帐户类型对应的同步适配器应在联系人提供程序与服务器之间传送组数据。 当用户向服务器添加一个新联系人，然后将该联系人放入一个新组时，同步适配器必须将这个新组添加到 `ContactsContract.Groups` 表中。 原始联系人所属的一个或多个组使用 `ContactsContract.CommonDataKinds.GroupMembership` MIME 类型存储在 `ContactsContract.Data` 表内。

如果您设计的同步适配器会将服务器中的原始联系人数据添加到联系人提供程序，并且您不使用组，则需要指示提供程序让您的数据可见。 在用户向设备添加帐户时执行的代码中，更新联系人提供程序为该帐户添加的 `ContactsContract.Settings` 行。 在该行中，将 `Settings.UNGROUPED_VISIBLE` 列的值设置为 1。执行此操作后，即使您不使用组，联系人提供程序也会让您的联系人数据始终可见。

联系人照片

`ContactsContract.Data` 表通过 MIME 类型 `Photo.CONTENT_ITEM_TYPE` 以行的形式存储照片。该行的 `CONTACT_ID` 列链接到其所属原始联系人的 `_ID` 列。 `ContactsContract.Contacts.Photo` 类定义了一个 `ContactsContract.Contacts` 子表，其中包含联系人主要照片（联系人的主要原始联系人的主要照片）的照片信息。 同样， `ContactsContract.RawContacts.DisplayPhoto` 类定义了一个 `ContactsContract.RawContacts` 子表，其中包含原始联系人主要照片的照片信息。

`ContactsContract.Contacts.Photo` 和 `ContactsContract.RawContacts.DisplayPhoto` 参考文档包含检索照片信息的示例。 并没有可用来检索原始联系人主要缩略图的实用类，但您可以向 `ContactsContract.Data` 表发送查询，从而通过选定原始联系人的 `_ID`、 `Photo.CONTENT_ITEM_TYPE` 以及 `IS_PRIMARY` 列，找到原始联系人的主要照片行。

联系人的社交流数据也可能包含照片。这些照片存储在 android.provider.ContactsContract.StreamItemPhotos 表中，社交流照片部分对该表做了更详尽的描述。