日历提供程序

本文内容

关键类

- `CalendarContract.Calendars`
- `CalendarContract.Events`
- `CalendarContract.Attendees`
- `CalendarContract.Reminders`

日历提供程序是用户日历事件的存储区。您可以利用 Calendar Provider API 对日历、事件、参加者、提醒等执行查询、插入、更新和删除操作。

Calender Provider API 可供应用和同步适配器使用。规则因进行调用的程序类型而异。 本文主要侧重于介绍使用 Calendar Provider API 作为应用的情况。如需了解对各类同步适配器差异的阐述，请参阅同步适配器。

正常情况下，要想读取或写入日历数据，应用的清单文件必须包括用户权限中所述的适当权限。 为简化常见操作的执行，日历提供程序提供了一组 Intent，日历 Intent中对这些 Intent 做了说明。 这些 Intent 会将用户转到日历应用，执行插入事件、查看事件和编辑事件。 用户与日历应用交互，然后返回原来的应用。 因此，您的应用不需要请求权限，也不需要提供用于查看事件或创建事件的用户界面。

基础知识

内容提供程序存储数据并使其可供应用访问。 Android 平台提供的内容提供程序（包括日历提供程序）通常以一组基于关系型数据库模型的表格形式公开数据，在这个表格中，每一行都是一条记录，每一列都是特定类型和含义的数据。 应用和同步适配器可以通过 Calendar Provider API 获得对储存用户日历数据的数据库表的读取/写入权限。

每一个内容提供程序都会公开一个对其数据集进行唯一标识的公共 URI（包装成一个 `Uri` 对象）。 控制多个数据集（多个表）的内容提供程序会为每个数据集公开单独的 URI。 所有提供程序 URI 都以字符串“content://”开头。 这表示数据受内容提供程序的控制。 日历提供程序会为其每个类（表）定义 URI 常量。这些 URI 的格式为 `<class>.CONTENT_URI`。例如，`Events.CONTENT_URI`。

图 1 是对日历提供程序数据模型的图形化表示。它显示了将彼此链接在一起的主要表和字段。

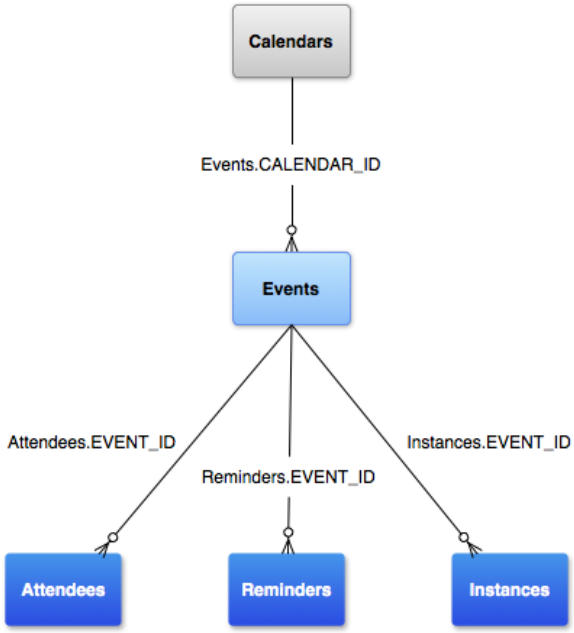

图 1. 日历提供程序数据模型。

用户可以有多个日历，可将不同类型的日历与不同类型的帐户（Google 日历、Exchange 等）关联。

`CalendarContract` 定义了日历和事件相关信息的数据模型。这些数据存储在以下所列的若干表中。

表（类）	说明
`CalendarContract.Calendars`	此表储存日历特定信息。 此表中的每一行都包含一个日历的详细信息，例如名称、颜色、同步信息等。
`CalendarContract.Events`	此表储存事件特定信息。 此表中的每一行都包含一个事件的信息 — 例如事件标题、地点、开始时间、结束时间等。 事件可一次性发生，也可多次重复发生。参加者、提醒和扩展属性存储在单独的表内。它们各自具有一个 `EVENT_ID`，用于引用 Events 表中的 `_ID`。
`CalendarContract.Instances`	此表储存每个事件实例的开始时间和结束时间。 此表中的每一行都表示一个事件实例。 对于一次性事件，实例与事件为 1:1 映射。对于重复事件，会自动生成多个行，分别对应多个事件实例。
`CalendarContract.Attendees`	此表储存事件参加者（来宾）信息。 每一行都表示事件的一位来宾。 它指定来宾的类型以及事件的来宾出席响应。
`CalendarContract.Reminders`	此表储存提醒/通知数据。 每一行都表示事件的一个提醒。一个事件可以有多个提醒。 每个事件的最大提醒数量在 `MAX_REMINDERS` 中指定，后者由拥有给定日历的同步适配器设置。 提醒以事件发生前的分钟数形式指定，其具有一个可决定用户提醒方式的方法。

Calendar Provider API 以灵活、强大为设计宗旨。提供良好的最终用户体验以及保护日历及其数据的完整性也同样重要。 因此，请在使用该 API 时牢记以下要点：

- **插入、更新和查看日历事件。**要想直接从日历提供程序插入事件、修改事件以及读取事件，您需要具备相应权限。不过，如果您开发的并不是完备的日历应用或同步适配器，则无需请求这些权限。您可以改用 Android 的日历应用支持的 Intent 将读取操作和写入操作转到该应用执行。当您使用 Intent 时，您的应用会将用户转到日历应用，在一个预填充表单中执行所需操作。 完成操作后，用户将返回您的应用。通过将您的应用设计为通过日历执行常见操作，可以为用户提供一致、可靠的用户界面。 这是推荐您采用的方法。 如需了解详细信息，请参阅日历 Intent。
- **同步适配器。**同步适配器用于将用户设备上的日历数据与其他服务器或数据源同步。 在 `CalendarContract.Calendars` 和 `CalendarContract.Events` 表中，预留了一些供同步适配器使用的列。提供程序和应用不应修改它们。 实际上，除非以同步适配器形式进行访问，否则它们处于不可见状态。 如需了解有关同步适配器的详细信息，请参阅同步适配器。

用户权限

如需读取日历数据，应用必须在其清单文件中加入 `READ_CALENDAR` 权限。文件中必须包括用于删除、插入或更新日历数据的 `WRITE_CALENDAR` 权限：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"...>
    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WRITE_CALENDAR" />
    ...
</manifest>
```

日历表

CalendarContract.Calendars 表包含各日历的详细信息。 应用和同步适配器均可写入下列日历列。如需查看所支持字段的完整列表，请参阅 CalendarContract.Calendars 参考资料。

常量	说明
NAME	日历的名称。
CALENDAR_DISPLAY_NAME	该日历显示给用户时使用的名称。
VISIBLE	表示是否选择显示该日历的布尔值。值为 0 表示不应显示与该日历关联的事件。 值为 1 表示应该显示与该日历关联的事件。此值影响 CalendarContract.Instances 表中行的生成。
SYNC_EVENTS	一个布尔值，表示是否应同步日历并将其事件存储在设备上。 值为 0 表示不同步该日历，也不将其事件存储在设备上。值为 1 表示同步该日历的事件，并将其事件存储在设备上。

查询日历

以下示例说明了如何获取特定用户拥有的日历。 为了简便起见，在此示例中，查询操作显示在用户界面线程（“主线程”）中。 实际上，此操作应该在一个异步线程而非主线程中完成。 如需查看更详细的介绍，请参阅加载器。 如果您的目的不只是读取数据，还要修改数据，请参阅 AsyncQueryHandler。

```
// Projection array. Creating indices for this array instead of doing
// dynamic lookups improves performance.
public static final String[] EVENT_PROJECTION = new String[] {
    Calendars._ID,                           // 0
    Calendars.ACCOUNT_NAME,                  // 1
    Calendars.CALENDAR_DISPLAY_NAME,         // 2
    Calendars.OWNER_ACCOUNT                  // 3
};

// The indices for the projection array above.
private static final int PROJECTION_ID_INDEX = 0;
private static final int PROJECTION_ACCOUNT_NAME_INDEX = 1;
private static final int PROJECTION_DISPLAY_NAME_INDEX = 2;
private static final int PROJECTION_OWNER_ACCOUNT_INDEX = 3;
```

在示例的下一部分，您需要构建查询。选定范围指定查询的条件。 在此示例中，查询寻找的是 ACCOUNT_NAME 为“sampleuser@google.com”、ACCOUNT_TYPE 为“com.google”、OWNER_ACCOUNT 为“sampleuser@google.com”的日历。如果您想查看用户查看过的所有日历，而不只是用户拥有的日历，请省略 OWNER_ACCOUNT。您可以利用查询返回的 Cursor 对象遍历数据库查询返回的结果集。 如需查看有关在内容提供程序中使用查询的详细介绍，请参阅内容提供程序。

```
// Run query
Cursor cur = null;
ContentResolver cr = getContentResolver();
Uri uri = Calendars.CONTENT_URI;
String selection = "((" + Calendars.ACCOUNT_NAME + " = ?) AND ("
                        + Calendars.ACCOUNT_TYPE + " = ?) AND ("
                        + Calendars.OWNER_ACCOUNT + " = ?))";
String[] selectionArgs = new String[] {"sampleuser@gmail.com", "com.google",
        "sampleuser@gmail.com"};
// Submit the query and get a Cursor object back.
cur = cr.query(uri, EVENT_PROJECTION, selection, selectionArgs, null);
```

以下后续部分使用游标单步调试结果集。它使用在示例开头设置的常量来返回每个字段的值。

> **为何必须加入 ACCOUNT_TYPE？**
>
> 如果您查询 Calendars.ACCOUNT_NAME，还必须将 Calendars.ACCOUNT_TYPE 加入选定范围。这是因为，对于给定帐户，只有在同时指定其 ACCOUNT_NAME 及其 ACCOUNT_TYPE 的情况下，才能将其视为唯一帐户。ACCOUNT_TYPE 字符串对应于在 AccountManager 处注册帐户时使用的帐户验证器。还有一种名为 ACCOUNT_TYPE_LOCAL 的特殊帐户类型，用于未关联设备帐户的日历。ACCOUNT_TYPE_LOCAL 帐户不会进行同步。

```
// Use the cursor to step through the returned records
while (cur.moveToNext()) {
    long calID = 0;
    String displayName = null;
    String accountName = null;
    String ownerName = null;

    // Get the field values
    calID = cur.getLong(PROJECTION_ID_INDEX);
    displayName = cur.getString(PROJECTION_DISPLAY_NAME_INDEX);
    accountName = cur.getString(PROJECTION_ACCOUNT_NAME_INDEX);
    ownerName = cur.getString(PROJECTION_OWNER_ACCOUNT_INDEX);

    // Do something with the values...

   ...
}
```

修改日历

如需执行日历更新，您可以通过 URI 追加 ID (`withAppendedId()`) 或第一个选定项形式提供日历的 `_ID`。 选定范围应以 `"_id=?"` 开头，并且第一个 `selectionArg` 应为事件的 `_ID`。您还可以通过在 URI 中编码 ID 来执行更新。 下例使用 (`withAppendedId()`) 方法更改日历的显示名称：

```
private static final String DEBUG_TAG = "MyActivity";
...
long calID = 2;
ContentValues values = new ContentValues();
// The new display name for the calendar
values.put(Calendars.CALENDAR_DISPLAY_NAME, "Trevor's Calendar");
Uri updateUri = ContentUris.withAppendedId(Calendars.CONTENT_URI, calID);
int rows = getContentResolver().update(updateUri, values, null, null);
Log.i(DEBUG_TAG, "Rows updated: " + rows);
```

插入日历

日历设计为主要由同步适配器进行管理，因此您只应以同步适配器形式插入新日历。 在大多数情况下，应用只能对日历进行一些表面更改，如更改显示名称。 如果应用需要创建本地日历，可以利用 `ACCOUNT_TYPE_LOCAL` 的 `ACCOUNT_TYPE`，通过以同步适配器形式执行日历插入来实现目的。`ACCOUNT_TYPE_LOCAL` 是一种特殊的帐户类型，用于未关联设备帐户的日历。 这种类型的日历不与服务器同步。如需了解对同步适配器的阐述，请参阅同步适配器。

事件表

`CalendarContract.Events` 表包含各事件的详细信息。要想添加、更新或删除事件，应用必须在其清单文件中加入 `WRITE_CALENDAR` 权限。

应用和同步适配器均可写入下列事件列。 如需查看所支持字段的完整列表，请参阅`CalendarContract.Events` 参考资料。

常量	说明
`CALENDAR_ID`	事件所属日历的 `_ID`。
`ORGANIZER`	事件组织者（所有者）的电子邮件。
`TITLE`	事件的标题。
`EVENT_LOCATION`	事件的发生地点。
`DESCRIPTION`	事件的描述。
`DTSTART`	事件开始时间，以从公元纪年开始计算的协调世界时毫秒数表示。
`DTEND`	事件结束时间，以从公元纪年开始计算的协调世界时毫秒数表示。
`EVENT_TIMEZONE`	事件的时区。
`EVENT_END_TIMEZONE`	事件结束时间的时区。
`DURATION`	RFC5545 格式的事件持续时间。例如，值为 `"PT1H"` 表示事件应持续一小时，值为 `"P2W"` 表示持续 2 周。
`ALL_DAY`	值为 1 表示此事件占用一整天（按照本地时区的定义）。 值为 0 表示它是常规事件，可在一天内的任何时间开始和结束。
`RRULE`	事件的重复发生规则格式。例如，`"FREQ=WEEKLY;COUNT=10;WKST=SU"`。 您可以在此处找到更多示例。
`RDATE`	事件的重复发生日期。 `RDATE` 与 `RRULE` 通常联合用于定义一组聚合重复实例。 如需查看更详细的介绍，请参阅

	RFC5545 规范。
AVAILABILITY	将此事件视为忙碌时间还是可调度的空闲时间。
GUESTS_CAN_MODIFY	来宾是否可修改事件。
GUESTS_CAN_INVITE_OTHERS	来宾是否可邀请其他来宾。
GUESTS_CAN_SEE_GUESTS	来宾是否可查看参加者列表。

添加事件

当您的应用插入新事件时，我们建议您按照使用 Intent 插入事件中所述使用 INSERT Intent。不过，您可以在需要时直接插入事件。 本节描述如何执行此操作。

以下是插入新事件的规则：

- 您必须加入 CALENDAR_ID 和 DTSTART。
- 您必须加入 EVENT_TIMEZONE。如需获取系统中已安装时区 ID 的列表，请使用 getAvailableIDs()。 请注意，如果您按使用 Intent 插入事件中所述通过 INSERT Intent 插入事件，则此规则不适用 — 在该情形下，系统会提供默认时区。
- 对于非重复事件，您必须加入 DTEND。
- 对于重复事件，您必须加入 DURATION 以及 RRULE 或 RDATE。请注意，如果您按使用 Intent 插入事件中所述通过 INSERT Intent 插入事件，则此规则不适用 — 在该情形下，您可以将 RRULE 与 DTSTART 和 DTEND 结合使用，日历应用会自动将其转换为持续时间。

以下是一个插入事件的示例。为了简便起见，此操作是在 UI 线程内执行的。实际上，应该在异步线程中完成插入和更新，以便将操作移入后台线程。 如需了解详细信息，请参阅 AsyncQueryHandler。

```
long calID = 3;
long startMillis = 0;
long endMillis = 0;
Calendar beginTime = Calendar.getInstance();
beginTime.set(2012, 9, 14, 7, 30);
startMillis = beginTime.getTimeInMillis();
Calendar endTime = Calendar.getInstance();
endTime.set(2012, 9, 14, 8, 45);
endMillis = endTime.getTimeInMillis();
...

ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Events.DTSTART, startMillis);
values.put(Events.DTEND, endMillis);
values.put(Events.TITLE, "Jazzercise");
values.put(Events.DESCRIPTION, "Group workout");
values.put(Events.CALENDAR_ID, calID);
values.put(Events.EVENT_TIMEZONE, "America/Los_Angeles");
Uri uri = cr.insert(Events.CONTENT_URI, values);

// get the event ID that is the last element in the Uri
long eventID = Long.parseLong(uri.getLastPathSegment());
//
// ... do something with event ID
//
//
```

> **注：**请注意以上示例如何在事件创建后捕获事件 ID。 这是获取事件 ID 的最简单方法。您经常需要使用事件 ID 来执行其他日历操作 — 例如，向事件添加参加者或提醒。

更新事件

当您的应用想允许用户编辑事件时，我们建议您按照使用 Intent 编辑事件中所述使用 EDIT Intent。不过，您可以在需要时直接编辑事件。 如需执行事件更新，您可以通过 URI 追加 ID (withAppendedId()) 或第一个选定项形式提供事件的 _ID。选定范围应以 "_id=?" 开头，并且第一个 selectionArg 应为事件的 _ID。 您还可以使用不含 ID 的选定范围执行更新。 以下是一个更新事件的示例。 它使用 withAppendedId() 方法更改事件的标题：

```
private static final String DEBUG_TAG = "MyActivity";
...
long eventID = 188;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
Uri updateUri = null;
// The new title for the event
values.put(Events.TITLE, "Kickboxing");
updateUri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
int rows = getContentResolver().update(updateUri, values, null, null);
Log.i(DEBUG_TAG, "Rows updated: " + rows);
```

删除事件

您可以通过将事件 `_ID` 作为 URI 追加 ID 或通过使用标准选定范围来删除事件。如果您使用追加 ID，则将无法同时使用选定范围。共有两个版本的删除：应用删除和同步适配器删除。应用删除将 *deleted* 列设置为 1。此标志告知同步适配器该行已删除，并且应将此删除操作传播至服务器。 同步适配器删除会将事件连同其所有关联数据从数据库中移除。 以下是一个应用通过事件 `_ID` 删除事件的示例：

```
private static final String DEBUG_TAG = "MyActivity";
...
long eventID = 201;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
Uri deleteUri = null;
deleteUri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
int rows = getContentResolver().delete(deleteUri, null, null);
Log.i(DEBUG_TAG, "Rows deleted: " + rows);
```

参加者表

`CalendarContract.Attendees` 表的每一行都表示事件的一位参加者或来宾。调用 `query()` 会返回一个参加者列表，其中包含具有给定 `EVENT_ID` 的事件的参加者。此 `EVENT_ID` 必须匹配特定事件的 `_ID`。

下表列出了可写入的字段。 插入新参加者时，您必须加入除 `ATTENDEE_NAME` 之外的所有字段。

常量	说明
`EVENT_ID`	事件的 ID。
`ATTENDEE_NAME`	参加者的姓名。
`ATTENDEE_EMAIL`	参加者的电子邮件地址。
`ATTENDEE_RELATIONSHIP`	参加者与事件的关系。下列值之一： • `RELATIONSHIP_ATTENDEE` • `RELATIONSHIP_NONE` • `RELATIONSHIP_ORGANIZER` • `RELATIONSHIP_PERFORMER` • `RELATIONSHIP_SPEAKER`
`ATTENDEE_TYPE`	参加者的类型。下列值之一： • `TYPE_REQUIRED` • `TYPE_OPTIONAL`
`ATTENDEE_STATUS`	参加者的出席状态。下列值之一： • `ATTENDEE_STATUS_ACCEPTED` • `ATTENDEE_STATUS_DECLINED` • `ATTENDEE_STATUS_INVITED` • `ATTENDEE_STATUS_NONE` • `ATTENDEE_STATUS_TENTATIVE`

添加参加者

以下是一个为事件添加一位参加者的示例。请注意，`EVENT_ID` 是必填项：

```
long eventID = 202;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Attendees.ATTENDEE_NAME, "Trevor");
values.put(Attendees.ATTENDEE_EMAIL, "trevor@example.com");
values.put(Attendees.ATTENDEE_RELATIONSHIP, Attendees.RELATIONSHIP_ATTENDEE);
values.put(Attendees.ATTENDEE_TYPE, Attendees.TYPE_OPTIONAL);
values.put(Attendees.ATTENDEE_STATUS, Attendees.ATTENDEE_STATUS_INVITED);
values.put(Attendees.EVENT_ID, eventID);
Uri uri = cr.insert(Attendees.CONTENT_URI, values);
```

提醒表

`CalendarContract.Reminders` 表的每一行都表示事件的一个提醒。调用 `query()` 会返回一个提醒列表，其中包含具有给定 `EVENT_ID` 的事件的提醒。

下表列出了提醒的可写入字段。插入新提醒时，必须加入所有字段。 请注意，同步适配器指定它们在 `CalendarContract.Calendars` 表中支持的提醒类型。 详情请参阅 `ALLOWED_REMINDERS`。

常量	说明
`EVENT_ID`	事件的 ID。
`MINUTES`	事件发生前的分钟数，应在达到该时间时发出提醒。
`METHOD`	服务器上设置的提醒方法。下列值之一： • `METHOD_ALERT` • `METHOD_DEFAULT` • `METHOD_EMAIL` • `METHOD_SMS`

添加提醒

下例显示如何为事件添加提醒。提醒在事件发生前 15 分钟发出。

```
long eventID = 221;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Reminders.MINUTES, 15);
values.put(Reminders.EVENT_ID, eventID);
values.put(Reminders.METHOD, Reminders.METHOD_ALERT);
Uri uri = cr.insert(Reminders.CONTENT_URI, values);
```

实例表

`CalendarContract.Instances` 表储存事件实例的开始时间和结束时间。 此表中的每一行都表示一个事件实例。 实例表无法写入，只提供查询事件实例的途径。

下表列出了一些您可以执行实例查询的字段。请注意，时区由 `KEY_TIMEZONE_TYPE` 和 `KEY_TIMEZONE_INSTANCES` 定义。

常量	说明
`BEGIN`	实例的开始时间，以协调世界时毫秒数表示。
`END`	实例的结束时间，以协调世界时毫秒数表示。
`END_DAY`	与日历时区相应的实例儒略历结束日。
`END_MINUTE`	从日历时区午夜开始计算的实例结束时间（分钟）。
`EVENT_ID`	该实例对应事件的 `_ID`。

START_DAY	与日历时区相应的实例儒略历开始日。
START_MINUTE	从日历时区午夜开始计算的实例开始时间（分钟）。

查询实例表

如需查询实例表，您需要在 URI 中指定查询的时间范围。 在以下示例中，CalendarContract.Instances 通过其 CalendarContract.EventsColumns 接口实现获得对 TITLE 字段的访问权限。换言之，TITLE 是通过数据库视图，而不是通过查询原始 CalendarContract.Instances 表返回的。

```
private static final String DEBUG_TAG = "MyActivity";
public static final String[] INSTANCE_PROJECTION = new String[] {
    Instances.EVENT_ID,      // 0
    Instances.BEGIN,         // 1
    Instances.TITLE          // 2
  };

// The indices for the projection array above.
private static final int PROJECTION_ID_INDEX = 0;
private static final int PROJECTION_BEGIN_INDEX = 1;
private static final int PROJECTION_TITLE_INDEX = 2;
...

// Specify the date range you want to search for recurring
// event instances
Calendar beginTime = Calendar.getInstance();
beginTime.set(2011, 9, 23, 8, 0);
long startMillis = beginTime.getTimeInMillis();
Calendar endTime = Calendar.getInstance();
endTime.set(2011, 10, 24, 8, 0);
long endMillis = endTime.getTimeInMillis();

Cursor cur = null;
ContentResolver cr = getContentResolver();

// The ID of the recurring event whose instances you are searching
// for in the Instances table
String selection = Instances.EVENT_ID + " = ?";
String[] selectionArgs = new String[] {"207"};

// Construct the query with the desired date range.
Uri.Builder builder = Instances.CONTENT_URI.buildUpon();
ContentUris.appendId(builder, startMillis);
ContentUris.appendId(builder, endMillis);

// Submit the query
cur =  cr.query(builder.build(),
    INSTANCE_PROJECTION,
    selection,
    selectionArgs,
    null);

while (cur.moveToNext()) {
    String title = null;
    long eventID = 0;
    long beginVal = 0;

    // Get the field values
    eventID = cur.getLong(PROJECTION_ID_INDEX);
    beginVal = cur.getLong(PROJECTION_BEGIN_INDEX);
    title = cur.getString(PROJECTION_TITLE_INDEX);

    // Do something with the values.
    Log.i(DEBUG_TAG, "Event:  " + title);
    Calendar calendar = Calendar.getInstance();
    calendar.setTimeInMillis(beginVal);
    DateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
    Log.i(DEBUG_TAG, "Date: " + formatter.format(calendar.getTime()));
    }
 }
```

日历 Intent

您的应用不需要读取和写入日历数据的权限。它可以改用 Android 的日历应用支持的 Intent 将读取和写入操作转到该应用执行。下表列出了日历提供程序支持的 Intent：

操作	URI	说明	Extra
`VIEW`	`content://com.android.calendar/time/<ms_since_epoch>` 您还可以通过 `CalendarContract.CONTENT_URI` 引用 URI。如需查看使用该 Intent 的示例，请参阅使用 Intent 查看日历数据。	打开日历后定位到 `<ms_since_epoch>` 指定的时间。	无。
`VIEW`	`content://com.android.calendar/events/<event_id>` 您还可以通过 `Events.CONTENT_URI` 引用 URI。如需查看使用该 Intent 的示例，请参阅使用 Intent 查看日历数据。	查看 `<event_id>` 指定的事件。	`CalendarContract.EXTRA_EVENT_BEGIN_TIME` `CalendarContract.EXTRA_EVENT_END_TIME`
`EDIT`	`content://com.android.calendar/events/<event_id>` 您还可以通过 `Events.CONTENT_URI` 引用 URI。如需查看使用该 Intent 的示例，请参阅使用 Intent 编辑事件。	编辑 `<event_id>` 指定的事件。	`CalendarContract.EXTRA_EVENT_BEGIN_TIME` `CalendarContract.EXTRA_EVENT_END_TIME`
`EDIT` `INSERT`	`content://com.android.calendar/events` 您还可以通过 `Events.CONTENT_URI` 引用 URI。如需查看使用该 Intent 的示例，请参阅使用 Intent 插入事件。	创建事件。	下表列出的任一 Extra。

下表列出了日历提供程序支持的 Intent Extra：

Intent Extra	说明
`Events.TITLE`	事件的名称。
`CalendarContract.EXTRA_EVENT_BEGIN_TIME`	事件开始时间，以从公元纪年开始计算的毫秒数表示。
`CalendarContract.EXTRA_EVENT_END_TIME`	事件结束时间，以从公元纪年开始计算的毫秒数表示。
`CalendarContract.EXTRA_EVENT_ALL_DAY`	一个布尔值，表示事件属于全天事件。值可以是 `true` 或 `false`。
`Events.EVENT_LOCATION`	事件的地点。
`Events.DESCRIPTION`	事件描述。
`Intent.EXTRA_EMAIL`	逗号分隔值形式的受邀者电子邮件地址列表。
`Events.RRULE`	事件的重复发生规则。
`Events.ACCESS_LEVEL`	事件是私人性质还是公共性质。
`Events.AVAILABILITY`	将此事件视为忙碌时间还是可调度的空闲时间。

下文描述如何使用这些 Intent。

使用 Intent 插入事件

您的应用可以利用 `INSERT` Intent 将事件插入任务转到日历应用执行。使用此方法时，您的应用甚至不需要在其清单文件中加入 `WRITE_CALENDAR` 权限。

当用户运行使用此方法的应用时，应用会将其转到日历来完成事件添加操作。 `INSERT` Intent 利用 extra 字段为表单预填充日历中事件的详细信息。 用户随后可取消事件、根据需要编辑表单或将事件保存到日历中。

以下是一个代码段，用于安排一个在 2012 年 1 月 19 日上午 7:30 开始、8:30 结束的事件。请注意该代码段中的以下内容：

- 它将 `Events.CONTENT_URI` 指定为 URI。
- 它使用 `CalendarContract.EXTRA_EVENT_BEGIN_TIME` 和 `CalendarContract.EXTRA_EVENT_END_TIME` extra 字段为表单预填充事件的时间。 这些时间的值必须以从公元纪年开始计算的协调世界时毫秒数表示。
- 它使用 `Intent.EXTRA_EMAIL` extra 字段提供以逗号分隔的受邀者电子邮件地址列表。

```
Calendar beginTime = Calendar.getInstance();
beginTime.set(2012, 0, 19, 7, 30);
Calendar endTime = Calendar.getInstance();
endTime.set(2012, 0, 19, 8, 30);
Intent intent = new Intent(Intent.ACTION_INSERT)
        .setData(Events.CONTENT_URI)
        .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, beginTime.getTimeInMillis())
        .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, endTime.getTimeInMillis())
        .putExtra(Events.TITLE, "Yoga")
        .putExtra(Events.DESCRIPTION, "Group class")
        .putExtra(Events.EVENT_LOCATION, "The gym")
        .putExtra(Events.AVAILABILITY, Events.AVAILABILITY_BUSY)
        .putExtra(Intent.EXTRA_EMAIL, "rowan@example.com,trevor@example.com");
startActivity(intent);
```

使用 Intent 编辑事件

您可以按更新事件中所述直接更新事件。但使用 `EDIT` Intent 可以让不具有事件编辑权限的应用将事件编辑操作转到日历应用执行。当用户在日历中完成事件编辑后，将会返回原来的应用。

以下是一个 Intent 的示例，它为指定事件设置新名称，并允许用户在日历中编辑事件。

```
long eventID = 208;
Uri uri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
Intent intent = new Intent(Intent.ACTION_EDIT)
    .setData(uri)
    .putExtra(Events.TITLE, "My New Title");
startActivity(intent);
```

使用 Intent 查看日历数据

日历提供程序提供了两种不同的 `VIEW` Intent 使用方法：

- 打开日历并定位到特定日期。
- 查看事件。

下例显示如何打开日历并定位到特定日期：

```
// A date-time specified in milliseconds since the epoch.
long startMillis;
...
Uri.Builder builder = CalendarContract.CONTENT_URI.buildUpon();
builder.appendPath("time");
ContentUris.appendId(builder, startMillis);
Intent intent = new Intent(Intent.ACTION_VIEW)
    .setData(builder.build());
startActivity(intent);
```

下例显示如何打开事件进行查看：

```
long eventID = 208;
...
Uri uri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
Intent intent = new Intent(Intent.ACTION_VIEW)
   .setData(uri);
startActivity(intent);
```

同步适配器

应用和同步适配器在访问日历提供程序的方式上只存在微小差异：

- 同步适配器需要通过将 `CALLER_IS_SYNCADAPTER` 设置为 `true` 来表明它是同步适配器。
- 同步适配器需要提供 `ACCOUNT_NAME` 和 `ACCOUNT_TYPE` 作为 URI 中的查询参数。
- 与应用或小部件相比，同步适配器拥有写入权限的列更多。 例如，应用只能修改日历的少数几种特性，例如其名称、显示名称、能见度设置以及是否同步日历。 相比之下，同步适配器不仅可以访问这些列，还能访问许多其他列，例如日历颜色、时区、访问级别、地点等等。不过，同步适配器受限于它指定的 `ACCOUNT_NAME` 和 `ACCOUNT_TYPE`。

您可以利用以下帮助程序方法返回供与同步适配器一起使用的 URI：

```
static Uri asSyncAdapter(Uri uri, String account, String accountType) {
   return uri.buildUpon()
       .appendQueryParameter(android.provider.CalendarContract.CALLER_IS_SYNCADAPTER,"true")
       .appendQueryParameter(Calendars.ACCOUNT_NAME, account)
       .appendQueryParameter(Calendars.ACCOUNT_TYPE, accountType).build();
}
```

如需查看同步适配器的实现示例（并非仅限与日历有关的实现），请参阅 SampleSyncAdapter。