Android 软件开发

Content providers

秦兴国

xgqin@guet.edu.cn

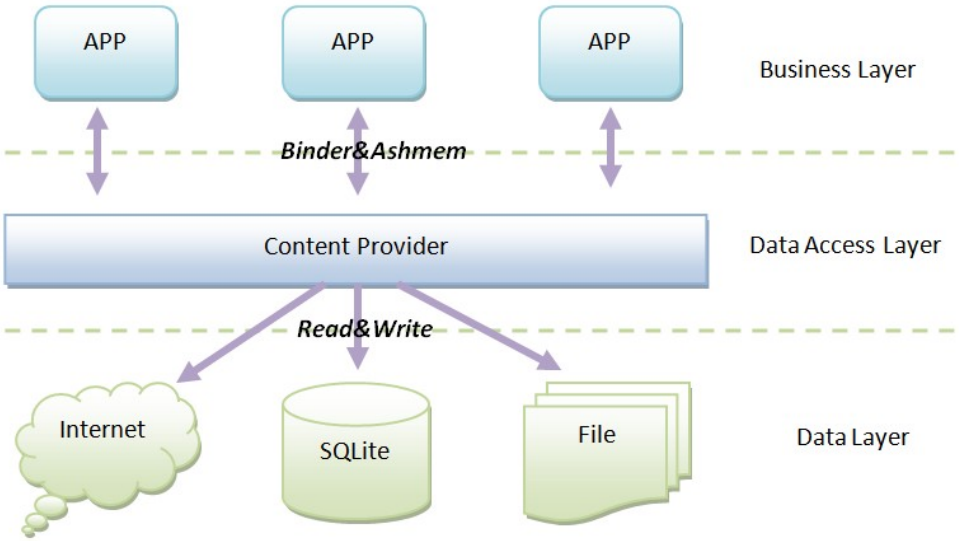
计算机与信息安全学院
桂林电子科技大学

2019 年 5 月 22 日

Outline

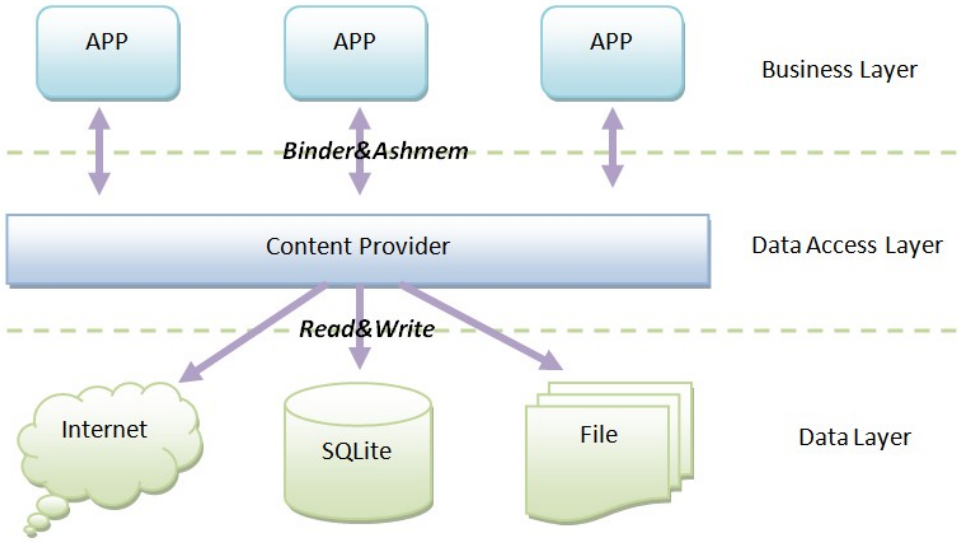
什么是 Content provider

Content provider 为共享数据提供访问接口，提供共享数据的应用可将数据存储与文件系统、SQLite 数据库或者是云端。



什么是 Content provider

Content provider 为共享数据提供访问接口，提供共享数据的应用可将数据存储与文件系统、SQLite 数据库或者是云端。

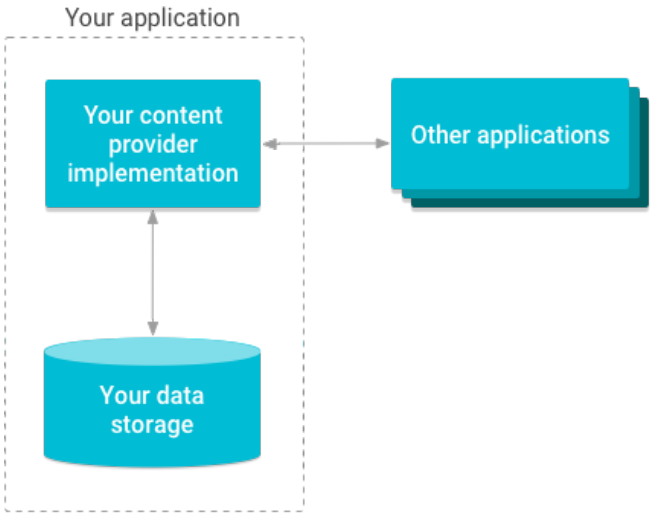


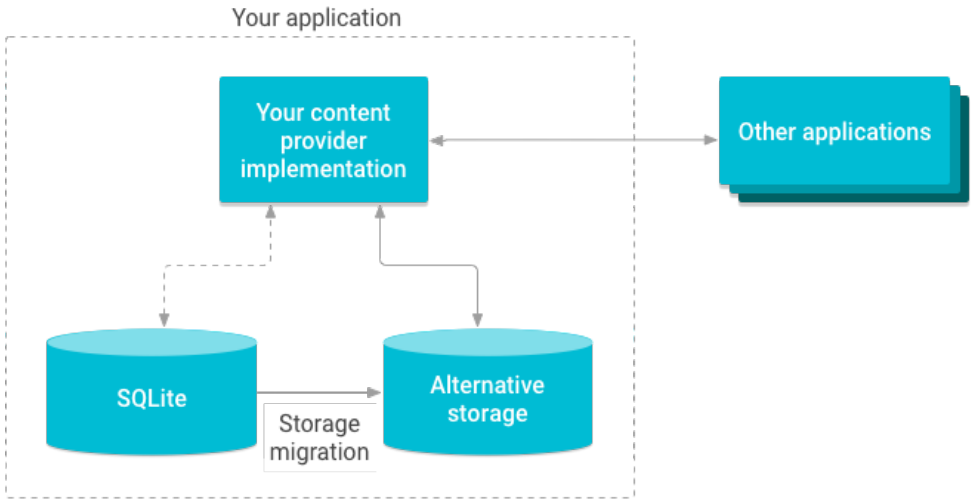
其他 App 通过 Content provider 提供的接口即可访问数据，而无需关心数据如何存储。

Content provider 的优点

Content provider 为数据跨进程间共享提供了标准的接口，其具备以下优点：

- Content provider 可通过访问权限确保数据访问安全；
- Content provider 提供统一的访问接口，为数据存储变更提供了向下兼容；





系统提供的 Content provider

Android 框架提供了众多 Content provider 为 App 访问系统数据：

- Contact provider，提供与联系人相关的数据操作接口；
- Media provider，提供与媒体资源（音频、视频、图片）相关的数据操作接口；
- Calendar provider，提供与日历相关的数据操作接口；
- CallLog provider，提供与通话记录相关的数据操作接口；
- Telephony provider，提供与 SMS、MMS 消息相关的数据操作接口；

何时需要自定义 Content provider

通常而言，App 通过访问 Android 框架提供 Content provider 访问系统共享数据，如果 App 需要对外提供数据，则需要自定义 Content provider：

- 在本应用中自定义搜索匹配；
- 向系统窗口 (Widget) 共享数据；
- 在应用建拷贝复杂的数据或文件；

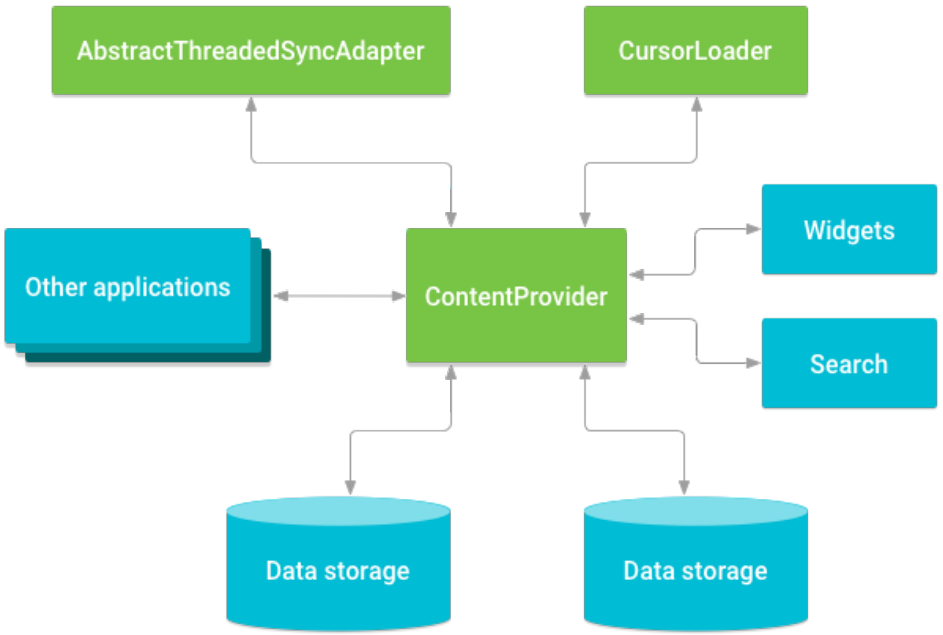
与 Content provider 相关的类

Android 框架提供了众多与 Content provider 配合使用的类库，包括：

- AbstractThreadedSyncAdapter，用于将本地 Content provider 数据与远端的服务进行同步；
- CursorAdapter，用于将 Content Resolver 查询到的 Cursor 数据与 AdapterView 类型控件进行绑定；
- CursorLoader，通过 Loader 异步框架查询 Content provider 数据；

Content provider 数据访问形式

Content provider 提供的数据通常以表格形式[1]对外部 App 进行呈现。Content provider 通过一系列的 API 接口向不同对象（远端服务、本地应用、第三方应用）提供数据访问接口。



[1] 这些表格概念与关系型数据库的数据表相似。表格中每一行数据表示的是 Content provider 提供的某一类型实例，每一列则表示该实例的某一属性。

使用 ContentResolver 访问 Content provider

Android 框架提供了 ContentResolver 类用于与 Content provider 进行数据访问操作。App 通过 ContentResolver 形式与 Content provider 进行解耦。

使用 ContentResolver 访问 Content provider

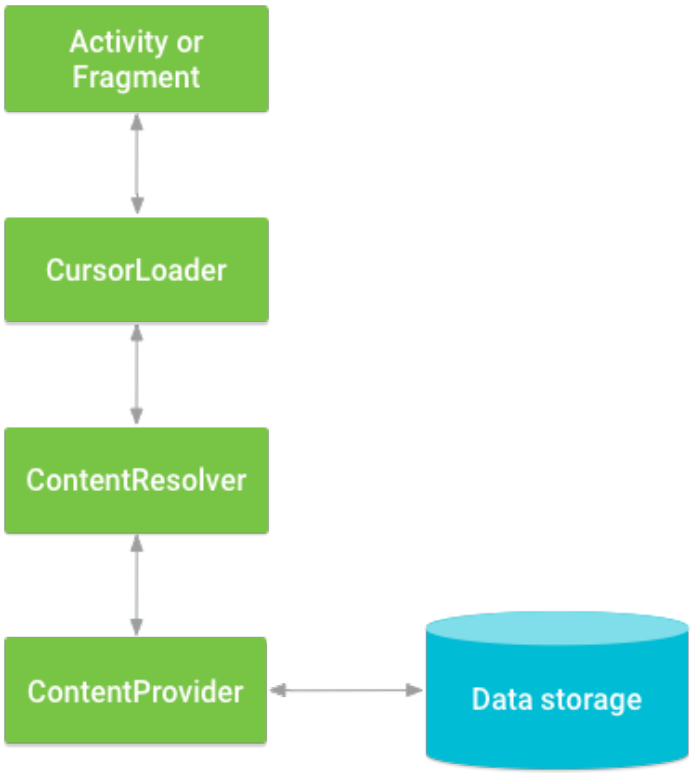
Android 框架提供了 ContentResolver 类用于与 Content provider 进行数据访问操作。App 通过 ContentResolver 形式与 Content provider 进行解耦。

ContentResolver 内部根据 App 提供的 Uri 实例化相应的 Content provider 对象，该对象接收从 ContentResolver 传递过来的数据请求，并执行请求操作返回操作结果。[2]

[2] ContentResolver 类提供了与 Contact provider 同名的方法，App 通过调用 ContentResolver 的 “CRUD” 方法对 Content provider 的数据进行访问。

使用 ContentResolver 访问 Content provider

Android 框架提供了 ContentResolver 类用于与 Content provider 进行数据访问操作。App 通过 ContentResolver 形式与 Content provider 进行解耦。



使用 ContentResolver 访问 Content provider 数据的场景：

1. 通过 CursorLoader 进行异步访问查询；
2. CursorLoader 根据 Uri 参数调用 ContentResolver 执行查询；
3. ContentResolver 根据 Uir 确定具体的 Content provider 类传递参数，获取结果。
4. CursorLoader 返回查询结果对应的 Cursor 对象，并由 App 决定如何使用 Cursor 对象访问数据。

使用 ContentResolver 访问 Content provider

用户字典是 Android 框架内建的 Content provider，它用于存储用户输入的非标准单词。其包含的数据表如下所示：

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	13
const	user1	225	pt_BR	4
int	user5	100	en_UK	5

使用 ContentResolver 访问 Content provider

```
1 // Queries the user dictionary and returns results
2 cursor = getContentResolver().query(
3     UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
4     projection,                         // The columns to return for each row
5     selectionClause,                    // Selection criteria
6     selectionArgs,                      // Selection criteria
7     sortOrder);                         // The sort order for the returned rows
```

使用 ContentResolver 访问 Content provider

```
1 // Queries the user dictionary and returns results
2 cursor = getContentResolver().query(
3     UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
4     projection,                         // The columns to return for each row
5     selectionClause,                    // Selection criteria
6     selectionArgs,                      // Selection criteria
7     sortOrder);                         // The sort order for the returned rows
```

通过 ContentResolver.query() 方法查询用户字典数据，该方法会调用用户字典的 Content provider 类对应的方法 ContentProvider.query()。

使用 ContentResolver 访问 Content provider

ContentResolver.query() 方法原型:

```
1  Cursor query(Uri uri, String[] projection,
2               String selection, String[] selectionArgs,
3               String sortOrder);
```

- Uri,
- projection,
- selection,
- selectionArgs,
- sortOrder,

使用 ContentResolver 访问 Content provider

Uri(Uniform Resource Identifier) 是形如以下形式的字符串标识[2]:

content://user_dictionary/words

[2] ContentResolver 通过 URI 解析出 Content provider 的符号名，从而确定所需实例化的 Content provider 类对象;

使用 ContentResolver 访问 Content provider

Uri(Uniform Resource Identifier) 是形如以下形式的字符串标识[2]:

content://user_dictionary/words

- content:// 称为 sheme，用于定义该 URI 的具体语法及关联的协议;
- user_dictionary 称为 Content provider 的 authority;
- words 则是某一数据表的 path;

[2] ContentResolver 通过 URI 解析出 Content provider 的符号名，从而确定所需实例化的 Content provider 类对象;

使用 ContentResolver 访问 Content provider

在 URI 的 path 后加上 *ID* 值表示访问 path 数据表对应的某一列值。

使用 ContentResolver 访问 Content provider

在 URI 的 path 后加上 *ID* 值表示访问 path 数据表对应的某一列值。例如：

content://user_dictionary/words/1

使用 ContentResolver 访问 Content provider

在 URI 的 path 后加上 *ID* 值表示访问 path 数据表对应的某一列值。例如：

content://user_dictionary/words/1

表示获取 *ID* 为 1 的用户字典单词 mapreduce 数据。

使用 ContentResolver 访问 Content provider

在 URI 的 path 后加上 *ID* 值表示访问 path 数据表对应的某一列值。例如：

content://user_dictionary/words/1

表示获取 *ID* 为 1 的用户字典单词 mapreduce 数据。

Uri、**Uri.Builder** 类提供了从字符串构造 URI 对象的方法。而 ContentUris 类则提供了在 URI 对象后追加 ID 值的方法。

使用 ContentResolver 访问 Content provider

在 URI 的 path 后加上 *ID* 值表示访问 path 数据表对应的某一列值。例如：

content://user_dictionary/words/1

表示获取 *ID* 为 1 的用户字典单词 mapreduce 数据。

Uri、Uri.Builder 类提供了从字符串构造 URI 对象的方法。而 ContentUris 类则提供了在 URI 对象后追加 ID 值的方法。

```
1       Uri singleUri = ContentUris.withAppendedId(
2                                   UserDictionary.Words.CONTENT_URI,
3                                   4);
```

声明 Content provider 的访问权限

Content provider 通常会定义数据读写访问权限，如果 Content provider 未指明访问权限，则：

- 其他 App 无法访问 Content provider 数据；
- Content provider 所在的同一应用的其他组件拥有完全访问权限。

声明 Content provider 的访问权限

Content provider 通常会定义数据读写访问权限，如果 Content provider 未指明访问权限，则：

- 其他 App 无法访问 Content provider 数据；
- Content provider 所在的同一应用的其他组件拥有完全访问权限。

Content provider 可设置 **android :grantUriPermissions** 属性对权限进行更细粒度的控制。

声明 Content provider 的访问权限

Content provider 的访问权限在应用的 AndroidManifest.xml 文件中对 Content provider 进行注册时声明：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      ...>
4
5      <application
6          ...>
7          <provider
8              android:name=".UserDictionary"
9              android:enabled="true"
10             android:exported="true"
11             android:readPermission="android.permission.READ_USER_DICTIONARY"
12             android:writePermission="android.permission.WRITE_USER_DICTIONARY"
13             ...
14             />
15     </application>
16 </manifest>
```

获取 Content provider 访问权限

App 在使用 Content provider 时需首先在 AndroidManifest.xml 文件中申请权限。

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.android.xgqin.providerdemo">
4
5     <uses-permission android:name="android.permission.READ_USER_DICTIONARY" />
6
7     ...
8
9 </manifest>
```

构造 Content provider 数据查询条件 I

通过 ContentResolver 构造查询条件:

```
1     String[] projection =
2     {
3         UserDictionary.Words._ID,
4         UserDictionary.Words.WORD,
5         UserDictionary.Words.LOCALE
6     };
7
8     String selectionClause = null;
9     String[] selectionArgs = {""};
10
11    searchString = searchWord.getText().toString();
12
13    if (TextUtils.isEmpty(searchString)) {
14        selectionClause = null;
15        selectionArgs[0] = "";
16    } else {
17        selectionClause = UserDictionary.Words.WORD + " = ?";
18        selectionArgs[0] = searchString;
```

构造 Content provider 数据查询条件 II

```
19      }
20
21      mCursor = getContentResolver().query(
22          UserDictionary.Words.CONTENT_URI,
23          projection,
24          selectionClause,
25          selectionArgs,
26          sortOrder);
27
28      if (null == mCursor) {
29          /*
30           * Insert code here to handle the error.
31           * You may want to
32           * call android.util.Log.e() to log this error.
33           *
34           */
35      // If the Cursor is empty, the provider found no matches
36      } else if (mCursor.getCount() < 1) {
37
38          /*
39           * Insert code here to notify the user that the search
40           * was unsuccessful. This isn't necessarily an error.
41           */
```

构造 Content provider 数据查询条件 III

```
42
43      } else {
44          // Insert code here to do something with the results
45      }
```

以上查询条件等价于：

```
SELECT _ID, word, locale FROM words
                    WHERE word = <userinput>
                    ORDER BY word ASC;
```

展示 Content provider 查询结果

ContentResolver.query() 方法始终返回 *Cursor* 对象，该对象包含了查询条件中 projection 指定的列数据。常见的做法是使用 CursorAdapter 将 Cursor 对象与 ListView 控件进行绑定：

展示 Content provider 查询结果

ContentResolver.query() 方法始终返回 *Cursor* 对象，该对象包含了查询条件中 projection 指定的列数据。常见的做法是使用 CursorAdapter 将 Cursor 对象与 ListView 控件进行绑定：

```
1     String[] wordListColumns = {
2         UserDictionary.Words.WORD,
3         UserDictionary.Words.LOCALE
4     };
5
6     int[] wordListItems = { R.id.dictWord, R.id.locale};
7
8     cursorAdapter = new SimpleCursorAdapter(
9         getApplicationContext(),
10        R.layout.wordlistrow,
11        mCursor,
12        wordListColumns,
13        wordListItems,
14        0);
15
16    wordList.setAdapter(cursorAdapter);
```

遍历 Content provider 查询结果

除了通过 CursorAdapter 将 Cursor 对象与 ListView 等控件绑定显示数据外，Cursor 对象还提供了随机访问查询结果的方法方便 App 遍历结果进行进一步的处理操作：

遍历 Content provider 查询结果

除了通过 CursorAdapter 将 Cursor 对象与 ListView 等控件绑定显示数据外，Cursor 对象还提供了随机访问查询结果的方法方便 App 遍历结果进行进一步的处理操作：

```
1     // Determine the column index of the column named "word"
2     int index = mCursor.getColumnIndex(UserDictionary.Words.WORD);
3
4     if (mCursor != null) {
5         while (mCursor.moveToNext()) {
6
7             // Gets the value from the column.
8             newWord = mCursor.getString(index);
9             ...
10        }
11    }
```

通过 ContentResolver 插入数据

向 Content provider 进行插入数据等更新操作同样需要使用 ContentResolver 对象。

调用 ContentResolver.insert() 方法向 Content provider 插入数据[3]，该方法返回的结果是新插入数据的 content URI：

```
1     Uri newUri;
2     ...
3     ContentValues newValues = new ContentValues();
4
5     newValues.put(UserDictionary.Words.APP_ID, "example.user");
6     newValues.put(UserDictionary.Words.LOCALE, "en_US");
7     newValues.put(UserDictionary.Words.WORD, "insert");
8     newValues.put(UserDictionary.Words.FREQUENCY, "100");
9
10    newUri = getContentResolver().insert(
11        UserDictionary.Words.CONTENT_URI,
12        newValues
13    );
```

[3] _ID 字段不需要用户指定，由 Content provider 进行维护，通常用该字段作为数据表的主键。

通过 ContentResolver 插入数据

```
1   Uri newUri;
2   ...
3   ContentValues newValues = new ContentValues();
4
5   newValues.put(UserDictionary.Words.APP_ID, "example.user");
6   newValues.put(UserDictionary.Words.LOCALE, "en_US");
7   newValues.put(UserDictionary.Words.WORD, "insert");
8   newValues.put(UserDictionary.Words.FREQUENCY, "100");
9
10  newUri = getContentResolver().insert(
11      UserDictionary.Words.CONTENT_URI,
12      newValues
13  );
```

ContentResolver.insert() 方法返回的 URI 为以下形式[3]：

content://user_dictionary/words/<id_value>

[3] 其中 <id_value> 为新插入行的 _ID 字段值，可以通过 ContentUris.parseId() 方法获取到该值。

通过 ContentResolver 更新数据

调用 ContentResolver.update() 方法更新 Content provider 的数据：

```
1     ContentValues updateValues = new ContentValues();
2
3     String selectionClause = UserDictionary.Words.LOCALE +  " LIKE ?";
4     String[] selectionArgs = {"en_%"};
5
6     int rowsUpdated = 0;
7
8     ...
9
10    updateValues.putNull(UserDictionary.Words.LOCALE);
11
12    rowsUpdated = getContentResolver().update(
13        UserDictionary.Words.CONTENT_URI,
14        updateValues,
15        selectionClause,
16        selectionArgs
17    );
```

通过 ContentResolver 删除数据

调用 ContentResolver.delete() 方法删除 Content provider 的数据：

```
1     String selectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
2     String[] selectionArgs = {"user"};
3
4     int rowsDeleted = 0;
5
6     ...
7
8     rowsDeleted = getContentResolver().delete(
9         UserDictionary.Words.CONTENT_URI,
10        selectionClause,
11        selectionArgs
12    );
```

通过 Intent 访问 Content provider

使用 Intent 对象访问 Content provider 的主要优点有：

- App 无需申请 Content provider 的访问权限；
- App 无需单独实现访问 Content provider 数据的代码及 UI 交互；

```
1    Intent pickIntent = new Intent(Intent.ACTION_PICK);
2    pickIntent.setType(ContactsContract.Contacts.CONTENT_TYPE);
3
4    if (pickIntent.resolveActivity(getPackageManager()) != null) {
5        startActivityForResult(pickIntent, PICK_CONTACT);
6    }
```

通过 Intent 访问 Content provider

使用 Intent 对象访问 Content provider 的主要优点有：

- App 无需申请 Content provider 的访问权限；
- App 无需单独实现访问 Content provider 数据的代码及 UI 交互；

```
1    Intent pickIntent = new Intent(Intent.ACTION_PICK);
2    pickIntent.setType(ContactsContract.Contacts.CONTENT_TYPE);
3
4    if (pickIntent.resolveActivity(getPackageManager()) != null) {
5        startActivityForResult(pickIntent, PICK_CONTACT);
6    }
```

通过 Intent 访问 Content provider

在 onActivityResult(int reqeustCode, int resultCode, Intent data) 方法内，返回响应 pickIntent 对象活动的结果，其中 data 对象包含了获取到的 Content provider 数据的 Uri 以及临时访问该 Uri 所需的权限。

```
1
2       @Override
3       protected void onActivityResult(int requestCode, int resultCode, Intent data) {
4           if (requestCode == PICK_CONTACT && resultCode == RESULT_OK) {
5               Uri contactUri = data.getData();
6               ...
7           }
8       }
```

Contact provider 简介

Contact provider 表结构，包含 Contact、Raw contact、Data 三个表可通过 contract 类进行引用：ContactContract.Contacts，ContactContract.RawContacts，ContactContract.Data，