西南交通大學
Southwest Jiaotong University

移动商务应用开发

第7章

ContentProvider与ContentResolver

学习内容

7.1 ContentProvider与ContentResolver概述

7.2 ContentProvider的使用

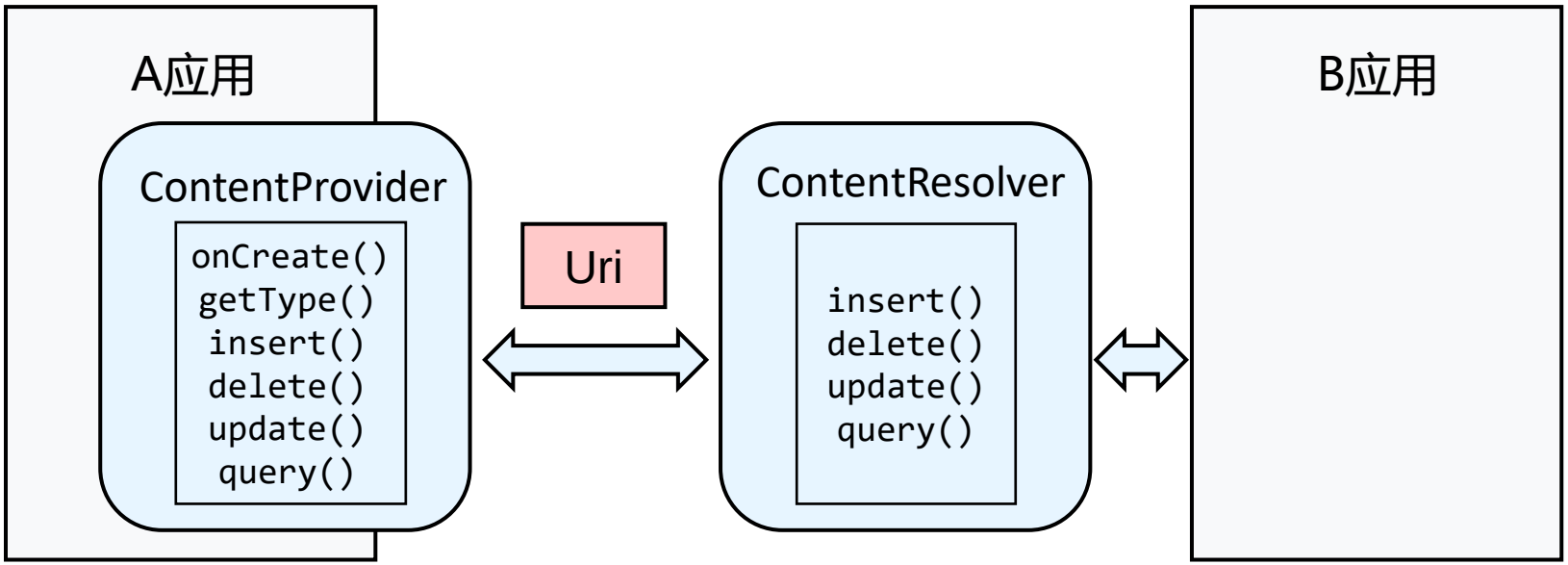
7.3 ContentResolver的使用

7.4 ContentObserver概述

7.5 ContentObserver的使用

ContentProvider与ContentResolver概述

- ContentProvider是Android的四大组件之一。
- ContentProvider是一个应用对外提供数据的接口，主要用于在不同应用间实现数据共享。



ContentProvider与ContentResolver概述

- ContentProvider与ContentResolver的工作原理：
  - A应用通过ContentProvider对外提供数据。
    - A应用需要创建自定义的ContentProvider类。
  - B应用通过ContentResolver访问其它应用的数据。
  - ContentProvider和ContentResolver提供了对数据进行增删改查的方法，二者的方法是一一对应的。
    - B应用调用ContentResolver的insert()时，Android系统会找到指定的应用并调用指定应用中的ContentProvider的insert()，所以用户无需显式调用ContentProvider的insert()。
  - ContentProvider和ContentResolver通过Uri来识别数据。

ContentProvider与ContentResolver概述

- Uri（统一资源标识符）为数据建立了唯一标识符。
- Uri由三部分组成：scheme、 authorities和path。
  - scheme指明了**协议**，content://是一个内容URI的标准协议。
  - authorities指明了是**哪个应用**，通常采用“包名.provider”的方式命名。
  - path指明了是**哪个数据**。

content:// com.example.room.provider /words

- 给定一个格式正确的字符串，可以通过Uri.parse()将字符串转换成Uri。

```
Uri uri = Uri.parse("com.example.room.provider/words")
```

学习内容

7.1 ContentProvider与ContentResolver概述

7.2 ContentProvider的使用

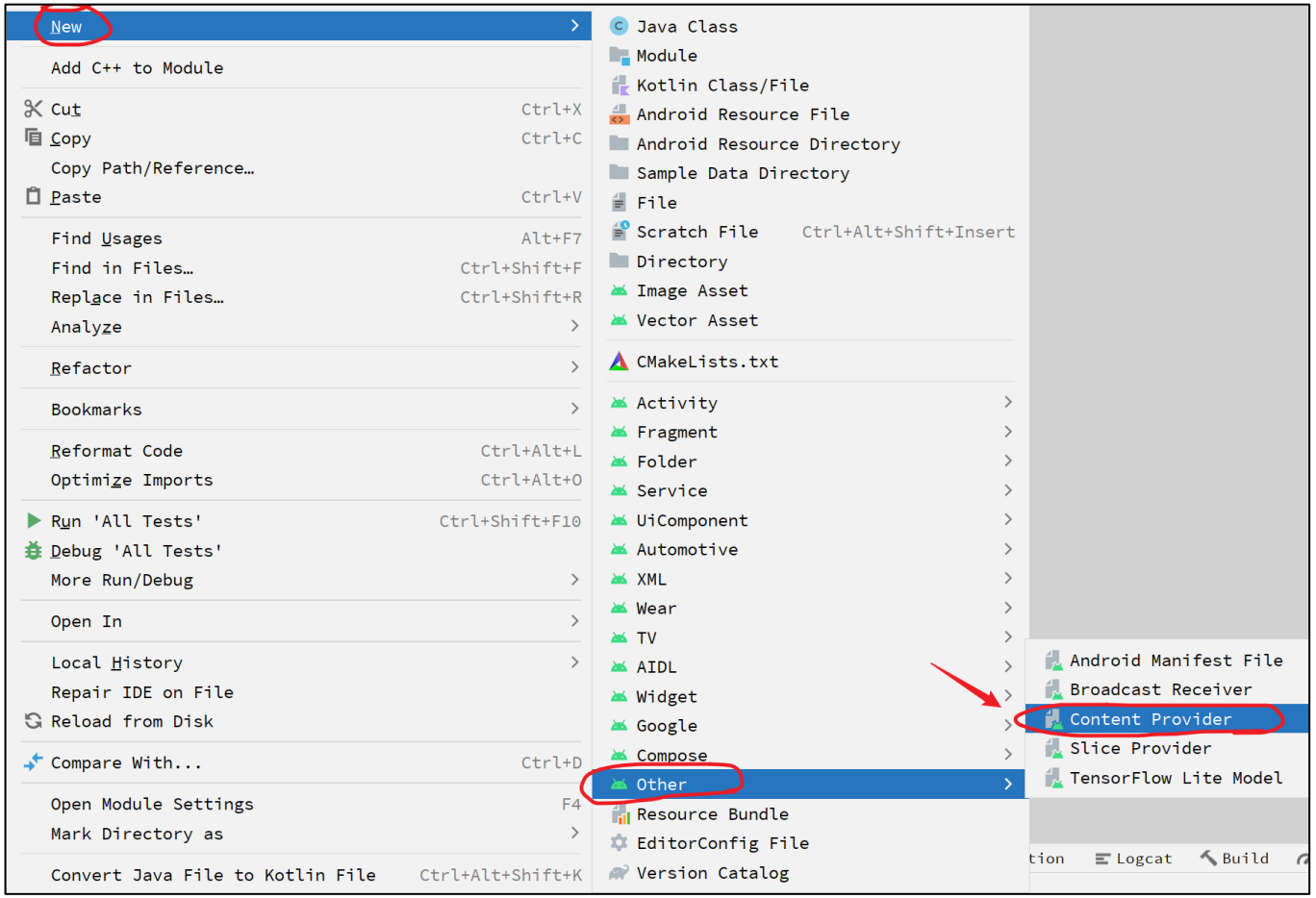
7.3 ContentResolver的使用

7.4 ContentObserver概述

7.5 ContentObserver的使用

创建ContentProvider

- A应用需要创建自定义的ContentProvider类。



创建ContentProvider

- Android Studio创建ContentProvider类的过程中完成了：
  - 创建了一个继承ContentProvider的java类
  - 将该ContentProvider在AndroidManifest.xml注册

```
<provider
      android:name=".MyContentProvider"
      android:authorities="com.example.room.provider"
      android:enabled="true"
      android:exported="true"></provider>
```

属性	含义
name	指定该ContentProvider类的名字
authorities	指定该ContentProvider对应的Uri
enabled	指定是否允许系统启动该ContentProvider
exported	指定该ContentProvider是否能被其它应用调用

指定ContentProvider中的内容URI

- A应用在创建ContentProvider类时需要指定内容Uri。
- Android提供了UriMatcher类来进行Uri的匹配，便于更简洁高效地管理Uri。
  - 利用UriMatcher类定义匹配规则

```
public static final String AUTHORITY = "com.example.contentprovider.provider";
public static final String TABLE = "words";
public static final int WORDS = 1;
// 利用UriMatcher类定义匹配规则
private static UriMatcher uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
static {
    uriMatcher.addURI(AUTHORITY, TABLE, WORDS);
}
```

静态方法块，用于初始化静态变量

将Uri和整型相匹配

表示匹配不成功的返回码

指定ContentProvider中的内容URI

- Android提供了UriMatcher类来进行Uri的匹配，便于更简洁高效地管理Uri。
  - 利用UriMatcher类根据匹配结果进行对应操作

```
switch (uriMatcher.match(uri)) {
    case WORDS:
        // 匹配成功后的对应操作
        break;
    default:
        break;
}
```

重写ContentProvider的方法

- A应用在创建ContentProvider类时需要重写以下方法：

方法	说明
boolean onCreate()	ContentProvider创建时被调用
String getType(Uri uri)	用于返回数据的类型
Uri insert(Uri uri, ContentValues values)	用于供外部应用往ContentProvider添加数据
int delete(Uri uri, String selection, String[] selectionArgs)	用于供外部应用从ContentProvider删除数据
int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)	用于供外部应用更新ContentProvider中的数据
Cursor query(Uri uri, String[] projection, String selection,String[] selectionArgs, String sortOrder)	用于供外部应用从ContentProvider中获取数据

重写ContentProvider的方法

boolean onCreate()	ContentProvider创建时被调用

- 当其它应用第一次访问该应用中的数据时，系统会自动创建ContentProvider实例，并调用onCreate()。
  - ContentProvider采用单例模式，即系统对A应用只会创建一个ContentProvider实例。
- 应用的数据大都以数据库的方式存储，因此在onCreate()中需要创建一个(Support)SQLiteOpenHelper实例，用于管理数据库。(Support)SQLiteOpenHelper是管理数据库的工具类。

重写ContentProvider的方法

boolean onCreate()	ContentProvider创建时被调用

```
private SupportSQLiteOpenHelper dbHelper;
public boolean onCreate() {
    // 获得SupportSQLiteOpenHelper
    WordDatabase wordDatabase = WordDatabase.getWordDatabase(getContext());
    dbHelper = wordDatabase.getOpenHelper();
    return true;
}
```

获得SupportSQLiteOpenHelper实例

重写ContentProvider的方法

String getType(Uri uri)	用于返回数据的类型

- 用于返回Uri指定的数据的类型，即MIME类型。
- 常见的MIME类型有：

扩展名	文档类型	MIME 类型
.csv	逗号分隔值（CSV）	text/csv
.doc	Microsoft Word	application/msword
.gif	图像互换格式（GIF）	image/gif
.htm, .html	超文本标记语言（HTML）	text/html
.jpeg, .jpg	JPEG 图像	image/jpeg
.mp3	MP3 音频	audio/mpeg

重写ContentProvider的方法

String getType(Uri uri)	用于返回数据的类型

- 也可为特殊的数据自定义其MIME类型。
  - 通常以“vnd.android.cursor.dir/xxx”或“vnd.android.cursor.item/xxx”的格式命名。

```
public static final String MIME_TYPE_WORDS = "vnd.android.cursor.dir/words";
public String getType(Uri uri) {
    switch (uriMatcher.match(uri)) {
        case WORDS:
            return MIME_TYPE_WORDS;
        default:
            return null;
    }
}
```

重写ContentProvider的方法

Uri insert(Uri uri, ContentValues values)	用于供外部应用往ContentProvider添加数据

- **Uri用于识别数据**
- **ContentValues values表示插入的内容，以键值对的方式存值**

```
public Uri insert(Uri uri, ContentValues values) {
    SupportSQLiteDatabase db = dbHelper.getWritableDatabase();
    switch (uriMatcher.match(uri)) {
        case WORDS:
            long id = db.insert(TABLE, OnConflictStrategy.IGNORE, values);
            Uri newUri = ContentUris.withAppendedId(uri, id);
            return newUri;
        default:
            return null;
    }
}
```

获得数据库

向TABLE插入数据

生成并返回新插入数据的Uri

重写ContentProvider的方法

```
long id = db.insert(TABLE, OnConflictStrategy.IGNORE, values);
```

- onConflictStrategy是Room提供的当存在主键冲突时的处理策略。
- 可选值包括：NONE, REPLACE, ROLLBACK, ABORT, FAIL, IGNORE
  - NONE表示不做任何操作，抛出异常（默认）
  - REPLACE表示替换掉旧数据
  - IGNORE表示忽略掉新数据

重写ContentProvider的方法

int delete(Uri uri, String selection, String[] selectionArgs)	用于供外部应用从ContentProvider删除数据

- **Uri用于识别数据**
- **String selection和String[] selectionArgs共同构成了筛选条件。**

Sqlite语句：

```
delete from words where english = `apple`;
```

SupportSQLiteDatabase的delete方法：

```
delete("words", "english = ?", new String[]{"apple"})
```

selection　　selectionArgs

重写ContentProvider的方法

int delete(Uri uri, String selection, String[] selectionArgs)	用于供外部应用从ContentProvider删除数据

- **Uri用于识别数据**
- **String selection和String[] selectionArgs**共同构成了筛选条件。

```
public int delete(Uri uri, String selection, String[] selectionArgs) {
    SupportSQLiteDatabase db = dbHelper.getWritableDatabase();
    switch (uriMatcher.match(uri)) {
        case WORDS:
            int v = db.delete(TABLE, selection, selectionArgs);
            return v;
        default:
            return -1;
    }
}
```

获得数据库

从TABLE中删除数据

重写ContentProvider的方法

int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)	用于供外部应用更新ContentProvider中的数据

- **Uri用于识别数据**
- **ContentValues values表示插入的内容，以键值对的方式存值**
- **String selection和String[] selectionArgs共同构成了筛选条件。**

```
public int update(Uri uri, ContentValues values, String selection,
                  String[] selectionArgs) {
    SupportSQLiteDatabase db = dbHelper.getWritableDatabase();
    switch (uriMatcher.match(uri)) {
        case WORDS:
            int v = db.update(TABLE, OnConflictStrategy.IGNORE, values,
                              selection, selectionArgs);
            return v;
        default:
            return -1;
    }
}
```

获得数据库

向TABLE更新数据

重写ContentProvider的方法

Cursor query(Uri uri, String[] projection, String selection,String[] selectionArgs, String sortOrder)	用于供外部应用从ContentProvider中获取数据

- **Uri用于识别数据**
- **String[] projection表示查看的字段**
- **String selection和String[] selectionArgs共同构成了筛选条件**
- **String sortOrder表示排序方式**

重写ContentProvider的方法

Cursor query(Uri uri, String[] projection, String selection,String[] selectionArgs, String sortOrder)	用于供外部应用从ContentProvider中获取数据

```
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {
    SupportSQLiteDatabase db = dbHelper.getWritableDatabase();
    switch (uriMatcher.match(uri)) {
        case WORDS:
            String sql = SupportSQLiteQueryBuilder.builder(TABLE)
                .columns(projection)
                .selection(selection, selectionArgs)
                .orderBy(sortOrder)
                .create().getSql();
            Cursor cursor = db.query(sql);
            return cursor;
        default:
            return null;
    }
}
```

获得数据库

查询TABLE中的数据

学习内容

7.1 ContentProvider与ContentResolver概述

7.2 ContentProvider的使用

7.3 ContentResolver的使用

7.4 ContentObserver概述

7.5 ContentObserver的使用

ContentResolver的使用

- B应用需要通过Context的getContentResolver()获取ContentResolver实例

```
ContentResolver contentResolver = getContentResolver();
```

- B应用可通过调用ContentResolver实例的方法来访问A应用的数据
  - insert()
  - delete()
  - update()
  - query()

ContentResolver的使用

- B应用的界面设计

ContentResolver的使用

- 通过ContentResolver访问words
  - 添加数据

```
void insertWord() {
    // 添加一条数据
    Uri uri = Uri.parse("content://com.example.contentprovider.provider/words");
    ContentValues values = new ContentValues();
    values.put("english", "pineapple");
    values.put("chinese", "菠萝");
    Uri newUri = contentResolver.insert(uri, values);
    String str = (newUri == null) ? "失败" : "成功";
    Log.d(TAG, "insertWord: " + str);
}
```

ContentResolver的使用

- 通过ContentResolver访问words
  - 删除数据

```
void deleteWord() {
    // 删除english = pineapple的数据
    Uri uri = Uri.parse("content://com.example.contentprovider.provider/words");
    String whereClause = "english = ?";
    String[] selectionArgs = new String[]{"pineapple"};
    int v = contentResolver.delete(uri, whereClause, selectionArgs);
    String str = (v == -1) ? "失败" : "成功";
    Log.d(TAG, "deleteWord: " + str);
}
```

ContentResolver的使用

- 通过ContentResolver访问words
  - 修改数据

```
void updateWord() {
    // 将english = pineapple的单词的中文修改为”凤梨“
    Uri uri = Uri.parse("content://com.example.contentprovider.provider/words");
    ContentValues values = new ContentValues();
    values.put("chinese", "凤梨");
    String whereClause = "english = ?";
    String[] selectionArgs = new String[]{"pineapple"};
    int v = contentResolver.update(uri, values, whereClause, selectionArgs);
    String str = (v == -1) ? "失败" : "成功";
    Log.d(TAG, "updateWord: " + str);
}
```

ContentResolver的使用

- 通过ContentResolver访问words
  - 查询数据

```
void queryWord() {
    // 查询所有单词
    Uri uri = Uri.parse("content://com.example.contentprovider.provider/words");
    Cursor cursor = contentResolver.query(uri, null, null, null, null);
    StringBuilder sb = new StringBuilder();
    // 遍历输出
    if (cursor != null) {
        while (cursor.moveToNext()) {
            int englishIndex = cursor.getColumnIndex("english"); // 0
            int chineseIndex = cursor.getColumnIndex("chinese"); // 1
            String english = cursor.getString(englishIndex);
            String chinese = cursor.getString(chineseIndex);
            sb.append(english + ", " + chinese + "\n");
        }
        cursor.close(); // 关闭cursor
        Log.d(TAG, "queryWord: \n" + sb);
    } else {
        Log.d(TAG, "queryWord: 查询失败");
    }
}
```

遍历输出数据

ContentResolver的使用

- 通过ContentResolver查询通讯录数据（系统应用）
  - 访问通讯录数据，首先需要静态授权+动态授权

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

```
// 动态授权
void getPermission() {
    String permission = Manifest.permission.READ_CONTACTS;
    if (ContextCompat.checkSelfPermission(this, permission)
        != PackageManager.PERMISSION_GRANTED) {
        requestPermissions(new String[]{permission}, 1);
    }
}
// 处理授权结果
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    switch (requestCode) {
        case 1:
            Toast.makeText(this, "授权成功", Toast.LENGTH_SHORT).show();
            break;
        default:
    }
}
```

ContentResolver的使用

- 通过ContentResolver查询通讯录数据

```
void queryContact() {
    // 查询所有联系人信息
    Uri uri = ContactsContract.CommonDataKinds.Phone.CONTENT_URI;
    Cursor cursor = contentResolver.query(uri, null, null, null, null);
    StringBuilder sb = new StringBuilder();
    // 遍历输出
    if (cursor != null) {
        while (cursor.moveToNext()) {
            int nameIndex =
cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME);
            int photoNumberIndex =
cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER);
            String name = cursor.getString(nameIndex);
            String photoNumber = cursor.getString(photoNumberIndex);
            sb.append(name + ", " + photoNumber + "\n");
        }
        cursor.close();
        Log.d(TAG, "queryContact: \n" + sb);
    } else {
        Log.d(TAG, "queryContact: 查询失败");
    }
}
```

系统定义的Uri，常量

系统定义的字符串，常量

学习内容

7.1 ContentProvider与ContentResolver概述

7.2 ContentProvider的使用

7.3 ContentResolver的使用

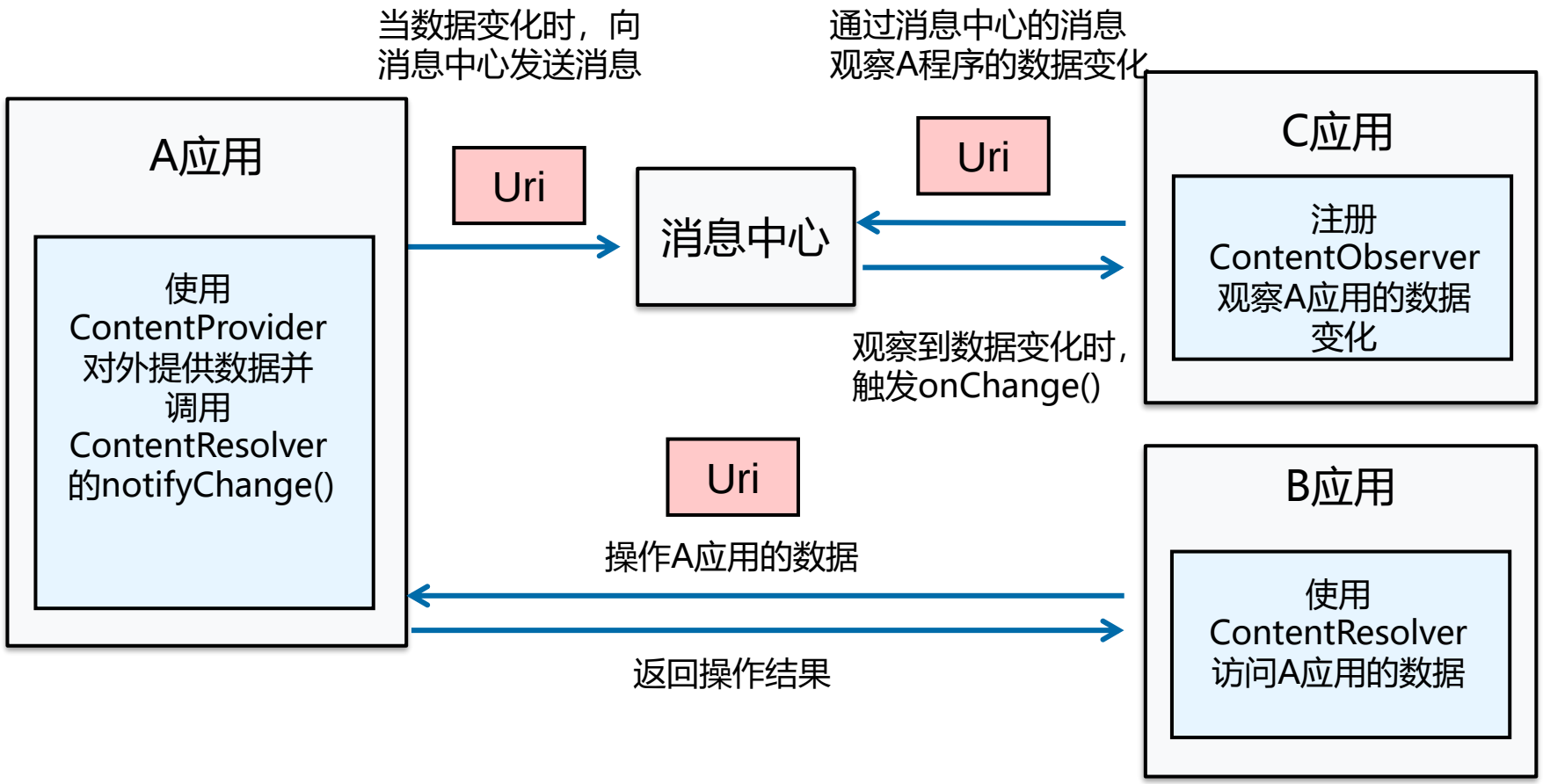
7.4 ContentObserver概述

7.5 ContentObserver的使用

ContentObserver概述

- ContentObserver主要用于观测由外部应用的操作导致的数据变化。
- 当ContentObserver观察到数据变化时，就会触发onChange()方法，此时在onChange()方法中可以自定义相关操作。
- 使用ContentObserver观察数据变化的步骤：
  - 在A应用的ContentProvider中调用ContentProvider中调用ContentResolver的notifyChange()
  - 在C应用中创建自定义的ContentObserver类，重写onChange()
  - 在C应用中注册ContentObserver

ContentObserver概述



学习内容

7.1 ContentProvider与ContentResolver概述

7.2 ContentProvider的使用

7.3 ContentResolver的使用

7.4 ContentObserver概述

7.5 ContentObserver的使用

ContentObserver的使用

- 在A应用的ContentProvider的增删改方法中调用ContentResolver的notifyChange()

```
public int delete(Uri uri, String selection, String[] selectionArgs) {
    SupportSQLiteDatabase db = dbHelper.getWritableDatabase();
    switch (uriMatcher.match(uri)) {
        case WORDS:
            int v = db.delete(TABLE, selection, selectionArgs);
            // 调用ContentResolver的notifyChange()从而通知所有注册在该URI上的ContentObserver
            getContext().getContentResolver().notifyChange(uri, null);
            Log.i(TAG, "delete: " + uri);
            return v;
        default:
            return -1;
    }
}
```

ContentObserver的使用

- 在C应用中创建自定义的ContentObserver类，并重写onChange()方法

```
public class MyContentObserver extends ContentObserver {
    private final String TAG = "myTagOnContentObserver";

    public MyContentObserver(Handler handler) {
        super(handler);
    }

    @Override
    public void onChange(boolean selfChange) {
        super.onChange(selfChange);
        Log.i(TAG, "监测数据变化：用外部应用正在改动words数据");
    }
}
```

ContentObserver的使用

- 在C应用中注册ContentObserver实例，观测特定Uri对应的数据。

```
// 通常写在onCreate()中
Uri uri = Uri.parse("content://com.example.contentprovider.provider/words");
MyContentObserver myContentObserver = new MyContentObserver(new Handler());
getContentResolver().registerContentObserver(uri, true, myContentObserver);
```

- 可以取消ContentObserver的注册

```
// 通常写在onDestroy()中
getContentResolver().unregisterContentObserver(myContentObserver);
```

主要内容

7.1 ContentProvider与ContentResolver概述

7.2 ContentProvider的使用

7.3 ContentResolver的使用

7.4 ContentObserver概述

7.5 ContentObserver的使用