Content Providers

Dominic Duggan
Stevens Institute of Technology

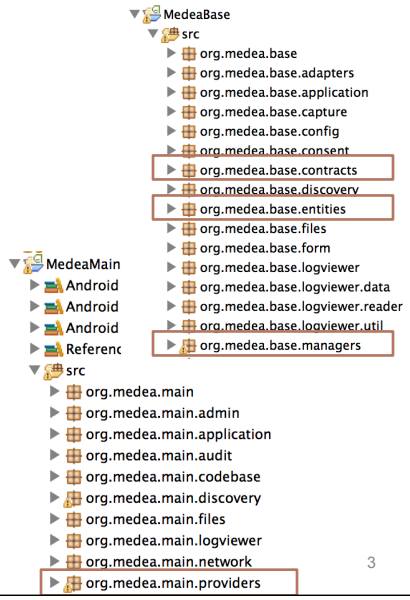
1

DEFINING A CONTENT PROVIDER

2

Defining a Content Provider

- Define contract class
  - `contracts` package
- Define entity class
  - `entities` package
- Define content provider
  - `providers` package
- Define manager class
  - `managers` package



CONTRACT CLASS

Contract Class

- Authority:
  - Extension of app namespace: "edu.stevens.cs522.bookstore"
- Content path:
  - Name of table: ("books") or item ("books/17")
- Content URI

```
CONTENT_URI(authority, path) =
    new Uri.Builder().scheme("content")
                     .authority(authority)
                     .path(path)
                     .build();

CONTENT_URI = CONTENT_URI("edu...", "books")
```

```
content://edu.stevens.cs522.bookstore/books
```

5

Contract Class

- Useful operations

```
public static Uri withExtendedPath(Uri uri,
                                   String... path) {
   Uri.Builder builder = uri.buildUpon();
   for (String p : path)
      builder.appendPath(p);
   return builder.build();
}

CONTENT_URI(id) = withExtendedPath(CONTENT_URI, id);

getId(uri) = Long.parseLong(uri.getLastPathSegment());
```

```
content://edu.stevens.cs522.bookstore/books/17
```

6

Contract Class

- Useful operations

```
CONTENT_PATH(Uri uri) = uri.getPath().substring(1);
                        // Trim leading "/”

CONTENT_PATH = CONTENT_PATH(CONTENT_URI);

CONTENT_PATH_ITEM = CONTENT_PATH(CONTENT_URI(“#”));
```

```
content://edu.stevens.cs522.bookstore/books
```

```
books
books/#
```

7

Contract Class

- Content types

```
contentType(content) =
    "vnd.android.cursor/vnd.”
           + APP_NAMESPACE + ".” + content + "s";
```

```
vnd.android.cursor/vnd.edu.stevens.cs522.bookstore.books
```

```
contentItemType(content) =
    "vnd.android.cursor.item/vnd.”
          + APP_NAMESPACE + ".” +content;
```

```
vnd.android.cursor.item/vnd.edu.stevens.cs522.bookstore.book
```

8

Contract Class

- Column identifiers:

```
public static final String TITLE = “title”;
public static final int TITLE_KEY = 1;
```

- Accessor operations for columns

```
public String getTitle(Cursor cursor) {
   int colIndex = cursor.getColumnIndexOrThrow(TITLE);
   return cursor.getString(colIndex);
}

public void putTitle(ContentValues values, String title){
   values.put(TITLE, title);
}
```

9

Contract Class

- Identity Column:

```
public static final String _ID = “_id”;
public static final int _ID_KEY = 0;
```

- Accessor operations for identity column:

```
public long getId(Cursor cursor) {
   int colIndex = cursor.getColumnIndexOrThrow(_ID);
   return cursor.getLong(colIndex);
}
```

- Rely on database insertion to set value for _ID

10

DEFINE ENTITY CLASS

11

Entity Class

- Define fields for entity
- Implement Parcelable
  - `Constructor(Parcel in)`
  - `void writeToParcel(Parcel out, int flags)`
- Accessor operations for provider
  - Use operations from contract
  - `Constructor(Cursor in)`
  - `void writeToProvider(ContentValues values)`

12

Entity Class: Example

```
public class Book implements Parcelable {
   public long id;
   public String title;

   public Book(Parcel in) { ... }
   public void writeToParcel(Parcel out) { ... }

   public Book(Cursor in) {
      this.id = BookContract.getId(in);
      this.title = BookContract.getString(in);
   }

   public void writeToProvider(ContentValues out) {
      BookContract.putTitle(this.title);
      ...
   }
}
```

13

DEFINE CONTENT PROVIDER

14

Defining a Content Provider

```
content://edu.stevens.cs522.bookstore/books/
content://edu.stevens.cs522.bookstore/books/17
```

```
public class BookProvider extends ContentProvider {


  @Override
  public boolean onCreate () {
    // TODO Construct the underlying database.
   return true;
  }

  // Create the constants used to differentiate
  // between the different URI  requests.
  private static final int ALL_ROWS = 1;
  private static final int SINGLE_ROW = 2;
```

15

Remember this?

- Useful contract operations

```
CONTENT_PATH(Uri uri) = uri.getPath().substring(1);
                        // Trim leading "/”

CONTENT_PATH = CONTENT_PATH(CONTENT_URI);

CONTENT_PATH_ITEM = CONTENT_PATH(CONTENT_URI(“#”));
```

```
content://edu.stevens.cs522.bookstore/books
```

```
books
books/#
```

16

Defining a Content Provider

```
content://edu.stevens.cs522.bookstore/books/
content://edu.stevens.cs522.bookstore/books/17
```

```
public class BookProvider extends ContentProvider {

  ...

  // Used to dispatch operation based on URI
  private static final UriMatcher uriMatcher;

  // uriMatcher.addURI(AUTHORITY, CONTENT_PATH, OPCODE)
  static {
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI(AUTHORITY,
                      CONTENT_PATH, ALL_ROWS);
    uriMatcher.addURI(AUTHORITY,
                      CONTENT_PATH_ITEM, SINGLE_ROW);
  }
```

books

books/#

17

Defining a Content Provider

```
content://edu.stevens.cs522.bookstore/books/
content://edu.stevens.cs522.bookstore/books/17
```

```
public class BookProvider extends ContentProvider {
  ...

 @Override
 public Cursor query (Uri uri, String[] projection,
                              String selection,
                              String[] selectionArgs,
                              String sort) {
   switch (uriMatcher.match(uri)) {
      case ALL_ROWS :
         // query the database

      case SINGLE_ROW :
         String selection = BookContract._ID + "=?";
         String[] selectionArgs = { getId(uri) };
         // query the database
   }
 }
```

18

Defining a Content Provider

```
content://edu.stevens.cs522.bookstore/books/
content://edu.stevens.cs522.bookstore/books/17
```

```
public class BookProvider extends ContentProvider {
  ...

 @Override
 public Cursor insert(Uri uri, ContentValues values) {
    long row = db.insert(TABLE_NAME, null, values);
    if (row > 0) {
       Uri instanceUri = BookContract.CONTENT_URI(row);

       ContentResolver cr =
          getContext().getContentResolver();
       cr.notifyChange(instanceUri, null);

       return instanceUri;
    }
    throw new SQLException("Insertion failed");
 }
```

19

Defining a Content Provider

```
content://edu.stevens.cs522.bookstore/books/
content://edu.stevens.cs522.bookstore/books/17
```

```
public class BookProvider extends ContentProvider {

  ...


  @Override
  public int delete (Uri uri,
                     String where,
                     String[] whereArgs) {
    switch (uriMatcher.match(uri)) {
      case ALL_ROWS:
      case SINGLE_ROW:
      default: throw new IllegalArgumentException
                              ("Unsupported URI:" + uri);
    }
  }
```

20

Defining a Content Provider

```
content://edu.stevens.cs522.bookstore/books/
content://edu.stevens.cs522.bookstore/books/17
```

```
public class BookProvider extends ContentProvider {

  ...


  @Override
  public int update (Uri uri,
                     ContentValues values,
                     String where,
                     String[] whereArgs) {
    switch (uriMatcher.match(uri)) {
      case ALL_ROWS:
      case SINGLE_ROW:
      default: throw new IllegalArgumentException
                          ("Unsupported URI:" + uri);
    }
  }
```

21

Defining a Content Provider

```
content://edu.stevens.cs522.bookstore/books/
content://edu.stevens.cs522.bookstore/books/17
```

```
public class BookProvider extends ContentProvider {

  ...


  @Override
  public String getType (Uri _uri) {
    switch (uriMatcher.match(_uri)) {
      case ALL_ROWS:
         return contentType("book");
      case SINGLE_ROW:
         return contentItemType("book");
      default: throw new IllegalArgumentException
                          ("Unsupported URI: " + _uri);
    }
  }
```

22

Defining a Content Provider

- Register the new content provider in the application manifest:

```
<provider
  android:name = ".providers.BookProvider"
  android:authorities =
      "edu.stevens.cs522.bookstore"
  android:exported = "false" />
```

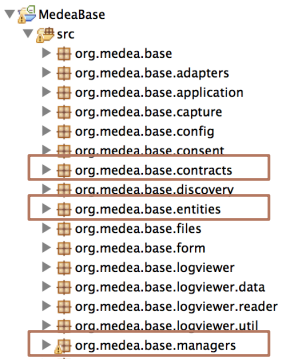
23

LOADING DATA

24

Best Practices

- Define entity classes
- Define contracts for providers
- Access provider via manager
- All accesses to 2ʸ storage are asynchronous
  - `CursorLoader` and `LoaderManager`
  - `AsyncQueryHandler`
  - `persist`, `persistAsync`, etc



25

How Not To Load Data

```
SimpleCursorAdapter adapter = null;

public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);

   // Set the layout
   setContentView(R.layout.cart);

   String[] projection = new String[] {
                            CartProvider._ID
                            CartProvider.TITLE,
                            CartProvider.AUTHOR
                         };
   Cursor c = managedQuery(CartProvider.CONTENT_URI,
                           projection, null, null, null);
   fillData(c);
}
```

26

```
private void fillData(Cursor c) {

   // Which layout object to bind to which data object.
   String[] to = new String[] { CartDbAdapter.TITLE,
                                CartDbAdapter.AUTHOR };
   int[] from = new int[] { R.id.cart_row_title,
                            R.id.cart_row_author };

   this.adapter = new SimpleCursorAdapter(
                this,     // Context.
                R.layout.cart_row,  // Row template
                c,        // Cursor encapsulates query result.
                to,       // Array of cursor columns to bind to.
                from);    // Parallel array of which layout objects
                          // to bind to those columns.

   ListView lv = (ListView)findViewById(android.R.id.list);
   lv.setAdapter(this.adapter);
}
```

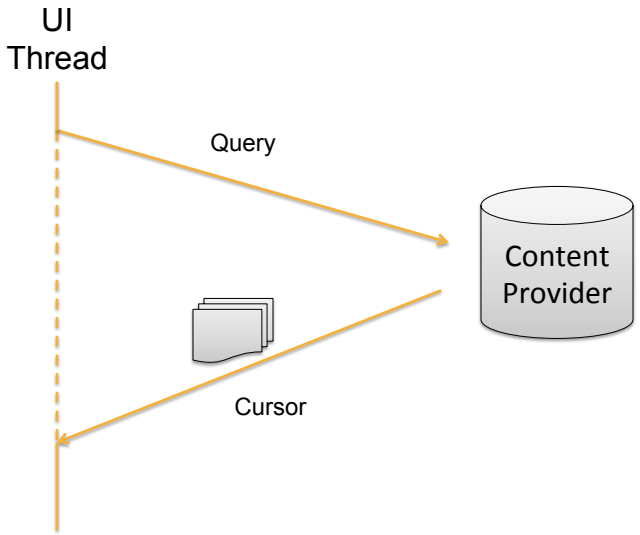
27

Problems

- managedQuery, startManagingCursor:
  - Re-queries database if activity is destroyed and recreated
- SimpleCursorAdapter constructor with cursor
  - Loads data on main UI thread

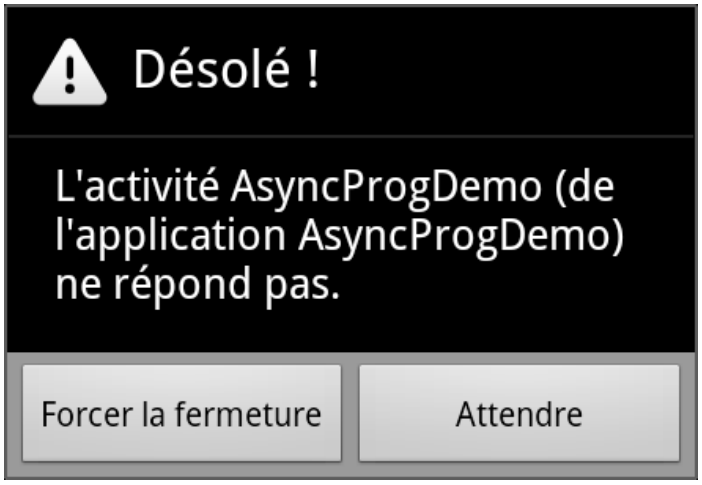
28

Problems



29

Do Not Let This Happen!

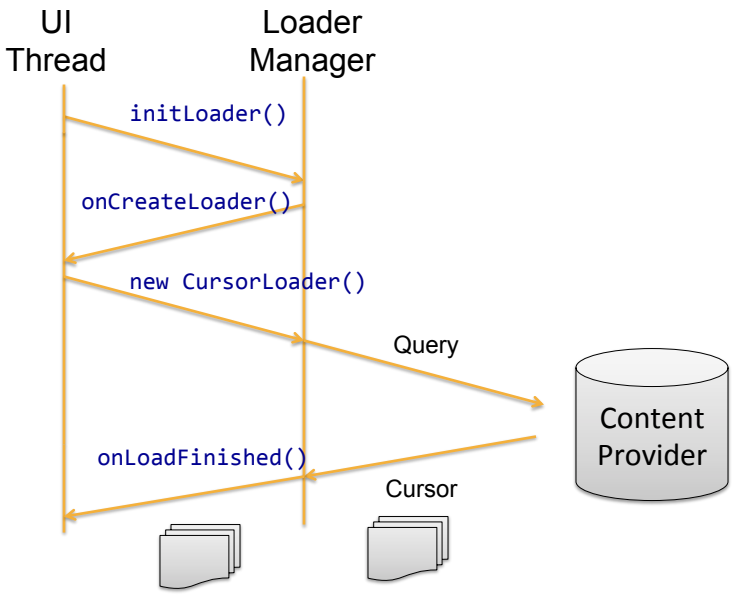


30

Solution: CursorLoader

- `CursorLoader`: Load data asynchronously in background
- `LoaderManager`: Manage cursor loaders (`LoaderManager.LoaderCallbacks<Cursor>`):
  - `onCreateLoader()`
  - `onLoadFinished()`

31



32

Create the Loader

- Implement the `LoaderCallbacks` interface
- Interact via callbacks with background loader

```
private static final int MY_LOADER_ID = 1;

public class MyActivity extends Activity
      implements LoaderManager.LoaderCallbacks<Cursor>{

public void onCreate(Bundle instanceState) {
   ...
   fillData(null);  // Initially no cursor

   LoaderManager lm = getLoaderManager();
   lm.initLoader(MY_LOADER_ID, null, this);
}
```

33

Start the Query

```
private static final int MY_LOADER_ID = 1;

public class MyActivity extends Activity
      implements LoaderManager.LoaderCallbacks<Cursor>{

   public Loader<Cursor> onCreateLoader(int loaderID,
                                        Bundle bundle) {
      switch (loaderID) {
        case MY_LOADER_ID:
           return new CursorLoader(this,
                           BookProvider.CONTENT_URI,
                           projection, null, null, null);
        default:
           return null; // An invalid id was passed in
    }}
```

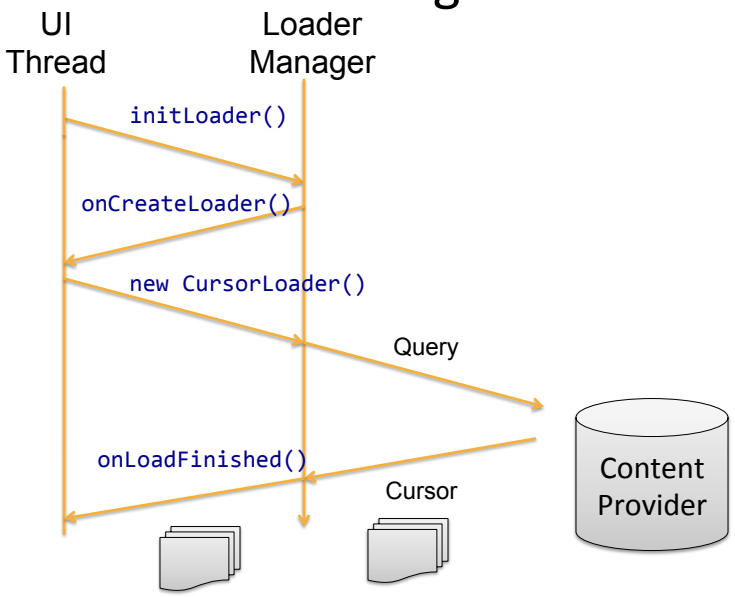
34

Respond to Query Completion

```
SimpleCursorAdapter adapter;

public class MyActivity extends Activity
      implements LoaderManager.LoaderCallbacks<Cursor>{

   public void onLoadFinished(Loader<Cursor> loader,
                              Cursor cursor) {
      this.adapter.swapCursor(cursor);
   }

   public void onLoaderReset(Loader<Cursor> loader) {
      this.adapter.swapCursor(null);
   }
}
```

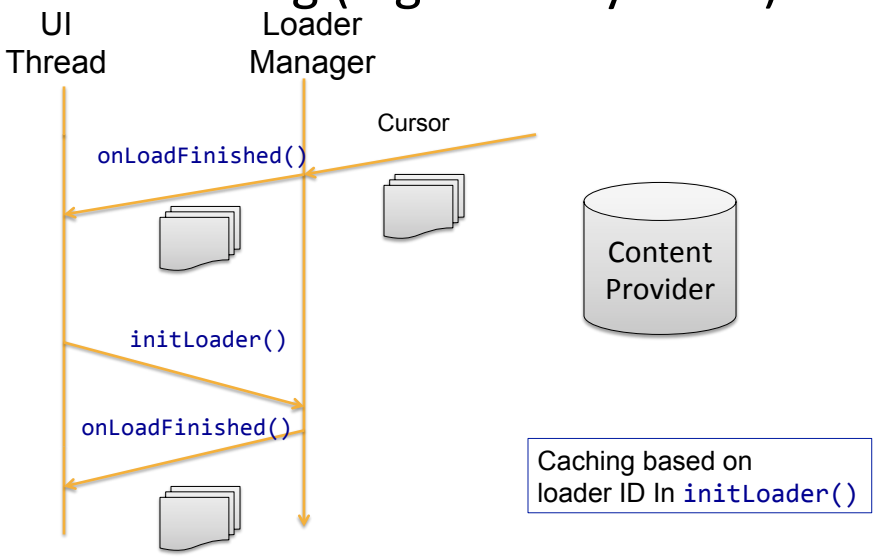
35

Starting Loader



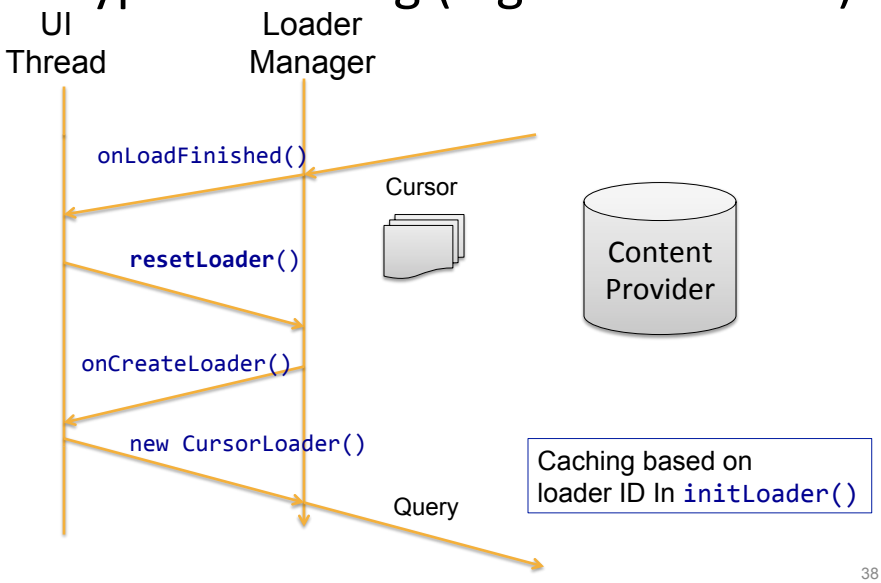
36

Caching (e.g. Activity reset)



37

Bypass Caching (e.g. new criteria)



38

Multiple Loaders

```
private static final int BOOKS_LOADER_ID = 1;
private static final int AUTHORS_LOADER_ID = 2;

public class MyActivity extends Activity
   implements LoaderManager.LoaderCallbacks<Cursor>{

public void onCreate(Bundle instanceState) {
   ...
   LoaderManager lm = getLoaderManager();
   lm.initLoader(BOOKS_LOADER_ID, null, this);
   lm.initLoader(AUTHORS_LOADER_ID, null, this);
   ...
}
```

39

Multiple Loaders

```
public class MyActivity extends Activity
      implements LoaderManager.LoaderCallbacks<Cursor>{

   public Loader<Cursor> onCreateLoader(int loaderID,
                                        Bundle bundle) {
      switch (loaderID) {
        case BOOKS_LOADER_ID:
           return new CursorLoader(...);
        case AUTHORS_LOADER_ID:
           return new CursorLoader(...);
        default:
           return null; // An invalid id was passed in
    }}
```

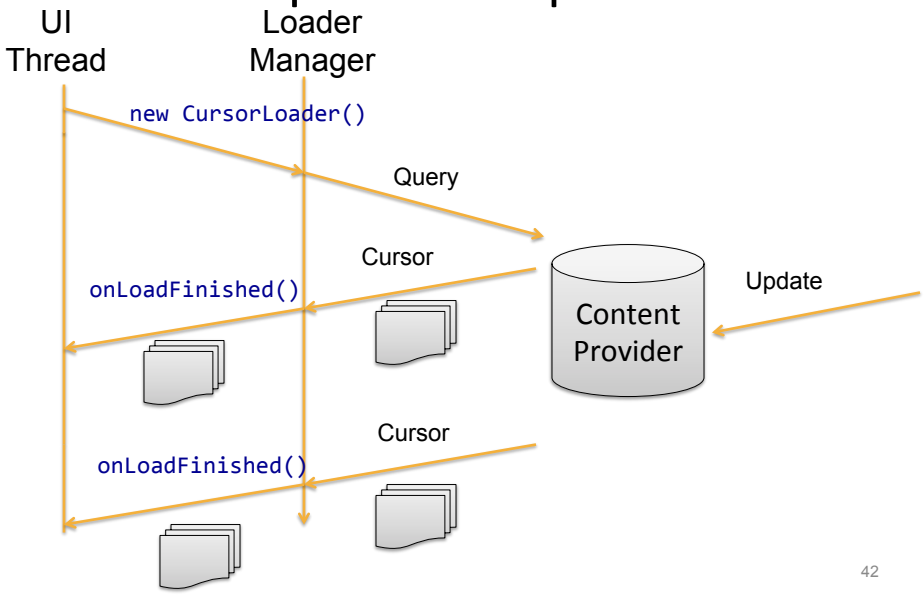
40

Multiple Loaders

```
SimpleCursorAdapter booksAdapter, authorsAdapter;

public class MyActivity extends Activity
      implements LoaderManager.LoaderCallbacks<Cursor>{

   public void onLoadFinished(Loader<Cursor> loader,
                              Cursor cursor) {
      switch(loader.getId()) {
      case BOOKS_LOADER_ID:
         this.booksAdapter.swapCursor(cursor);
         break;
      case AUTHORS_LOADER_ID:
         this.authorsAdapter.swapCursor(cursor);
         break;
   }
}
```

41

Response to Updates



42

Avoid This Scenario

- Example: Adding a book to a database
  - Read ISBN from UI
  - Query for ISBN already in database
  - If not found, insert into the database
  - (What happens next?)

43

LoaderManager Don’ts

- Don’t follow query with other work
- Don’t reuse loader ID (within an activity)
  - `onCreateLoader`: create book cursor
  - `onLoadFinished`: swap into author adapter
- Don’t close cursor
- Don’t perform UI changes on loader callback
  - More next time

44

Loader Manager: Concluding Notes

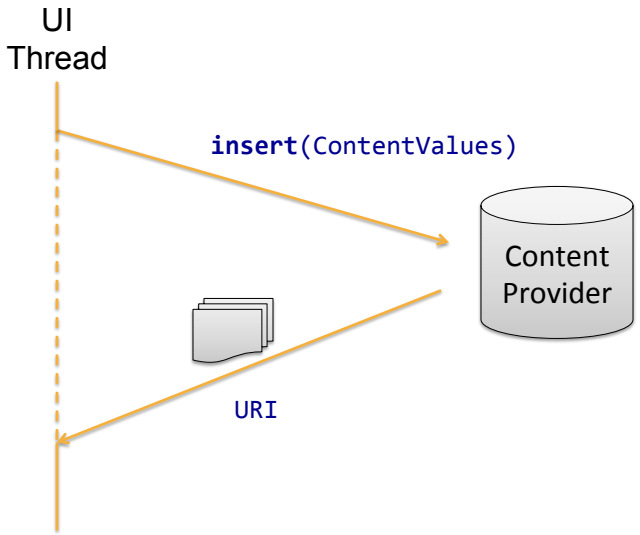
- Pro
  - Asynchronous querying
  - Caching (fast UI reset)
  - Refresh after database update
- Con:
  - Only querying
  - Bureaucracy
  - Don’t follow query with other work
- Purpose: Backend for list-driven UI

45

ASYNC QUERY HANDLER

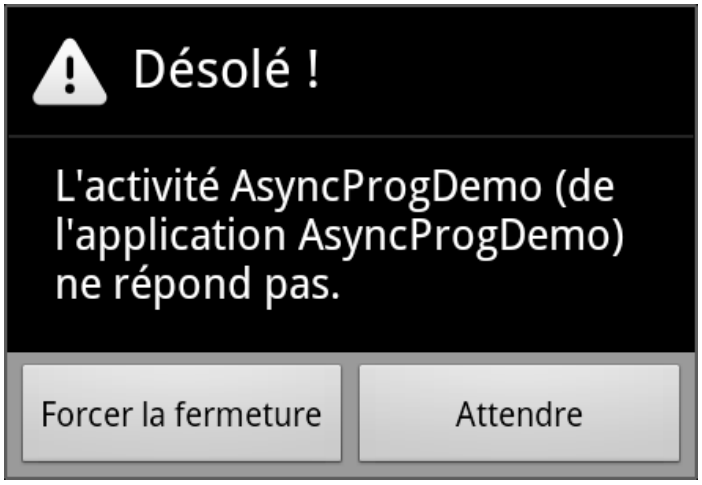
46

Content Resolver



47

Do Not Let This Happen!



48

AsyncQueryHandler

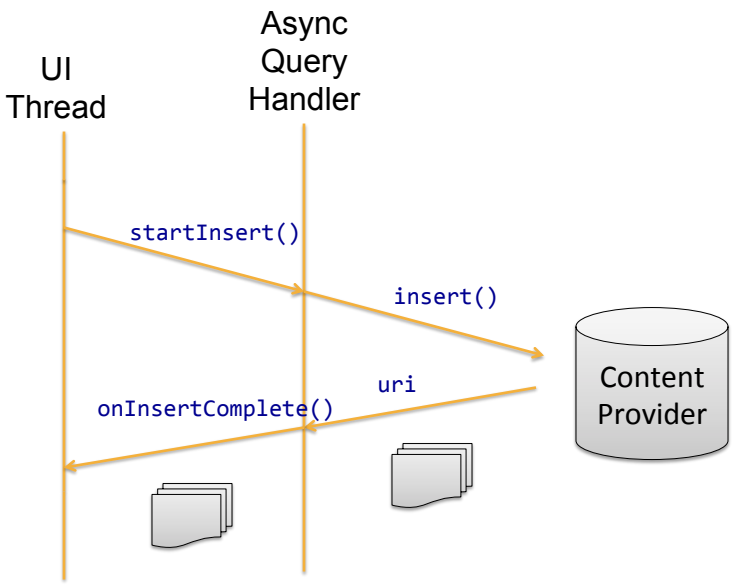
- Defines API to start async operations

```
public class AsyncQueryHandler {
   public void startInsert(..., ContentValues v);
}
```

- Subclass to add callbacks

```
public class MyContentResolver extends AsyncQueryHandler {
   @Override
   public void onInsertComplete(..., Uri uri) {
      ...
}
```

49



50

AsyncQueryHandler

Public Constructors	
	AsyncQueryHandler (ContentResolver cr)

Public Methods	
final void	cancelOperation (int token) Attempts to cancel operation that has not already started.
void	handleMessage (Message msg) Subclasses must implement this to receive messages.
final void	startDelete (int token, Object cookie, Uri uri, String selection, String[] selectionArgs) This method begins an asynchronous delete.
final void	startInsert (int token, Object cookie, Uri uri, ContentValues initialValues) This method begins an asynchronous insert.
void	startQuery (int token, Object cookie, Uri uri, String[] projection, String selection, String[] selectionArgs, String This method begins an asynchronous query.
final void	startUpdate (int token, Object cookie, Uri uri, ContentValues values, String selection, String[] selectionArgs) This method begins an asynchronous update.

51

AsyncQueryHandler

Protected Methods	
Handler	createHandler (Looper looper)
void	onDeleteComplete (int token, Object cookie, int result) Called when an asynchronous delete is completed.
void	onInsertComplete (int token, Object cookie, Uri uri) Called when an asynchronous insert is completed.
void	onQueryComplete (int token, Object cookie, Cursor cursor) Called when an asynchronous query is completed.
void	onUpdateComplete (int token, Object cookie, int result) Called when an asynchronous update is completed.

52

ASYNCHRONOUS CONTENT RESOLVER

53

Continuation API

- Define this interface yourself:

```
public interface IContinue<T> {
   public void kontinue(T value);
}
```

54

Continuation API

- Synchronous API:

```
Y foo(X arg) { ... return result; }
```

- Asynchronous API:

```
public interface IContinue<T> {
   public void kontinue(T value);
}
void fooAsync(X arg, IContinue<Y> callback) {
   ... callback.kontinue(result);
}
```

55

AsyncQueryHandler

```
public abstract class AsyncQueryHandler {

  public void startInsert(int token,
                          Object cookie,
                          Uri uri,
                          ContentValues values) {
    ...;
  }

  @Override
  public void abstract onInsertComplete(int token,
                                        Object cookie,
                                        Uri uri);
```

56

AsyncContentResolver

```
public class AsyncContentResolver extends AsyncQueryHandler {

  public void insertAsync(Uri uri,
                          ContentValues values,
                          IContinue<Uri> callback) {
    this.startInsert(0, callback, uri, values);
  }

  @Override
  public void onInsertComplete(int token, Object cookie, Uri uri) {
    if (cookie != null) {
      @SuppressWarnings("unchecked")
      IContinue<Uri> callback = (IContinue<Uri>) cookie;
      callback.kontinue(uri);
    }
  }
```

57

AsyncContentResolver Client

```
public class BookManager {

  AsyncContentResolver asyncResolver =
     new AsyncContentResolver(context.getContentResolver());

  final static String CONTENT_URI = BookContract.CONTENT_URI;

  public void persistAsync(Book book, IContinue<Uri> callback) {
     ContentValues values = new ContentValues();
     book.writeToProvider(values);
     asyncResolver.insertAsync(CONTENT_URI,
                               values,
                               callback);
  }
```

58

AsyncContentResolver Client

```
public class BookManager {

  AsyncContentResolver asyncResolver =
     new AsyncContentResolver(context.getContentResolver());

  final static String CONTENT_URI = BookContract.CONTENT_URI;

  public void persistAsync(final Book book) {
     ContentValues values = new ContentValues();
     book.writeToProvider(values);
     asyncResolver.insertAsync(CONTENT_URI,
                               values,
                               new IContinue<Uri>() {
                                  public void kontinue(Uri uri) {
                                     book.id = getId(uri);
                                  }
                               });
  }
```

59

AsyncContentResolver

```
public class AsyncContentResolver extends AsyncQueryHandler {

  public void queryAsync(Uri uri, String[] projection,
                         String selection, String[] selectionArgs,
                         String sortOrder,
                         IContinue<Cursor> callback) {
    this.startQuery(0, callback, uri, projection,
                    selection, selectionArgs, sortOrder);
  }

  @Override
  public void onQueryComplete(int token, Object cookie, Cursor c) {
    if (cookie != null) {
      @SuppressWarnings("unchecked")
      IContinue<Cursor> callback = (IContinue<Cursor>) cookie;
      callback.kontinue(c);
    }
  }
```

60

AsyncContentResolver

```
public void deleteAsync(Uri uri, String selection,
                        String[] selectionArgs) {
  this.startDelete(0, null, uri, selection, selectionArgs);
}

@Override
public void onDeleteComplete(int token, Object cookie, ...) {
}

public void updateAsync(Uri uri, ContentValues values,
                        String selection, String[] selectionArgs) {
  this.startUpdate(0, null, uri, values,
                        selection, selectionArgs);
}

@Override
public void onUpdateComplete(int token, Object cookie, ...) {
}
```

61

ASYNCHRONOUS QUERY FACTORIES

62

Entity Creator

- Interface for entity factory

```
public interface IEntityCreator<T> {

   public T create(Cursor cursor);

}
```

63

Simple Query Listener

- Callback for query results

```
public interface ISimpleQueryListener<T> {

   public void handleResults(List<T> results);

}
```

64

Simple Query

- Factory for queries

```
public class SimpleQueryBuilder<T>
                implements IContinue<Cursor> {

  private IEntityCreator<T> helper;
  private ISimpleQueryListener<T> listener;

  private SimpleQueryBuilder(
              IEntityCreator<T> helper,
              ISimpleQueryListener<T> listener) {
    this.helper = helper;
    this.listener = listener;
  }
```

65

Simple Query

- Factory for queries

```
public class SimpleQueryBuilder<T>
               implements IContinue<Cursor> {

  public static <T> void executeQuery(Activity context,
                    Uri uri,
                    IEntityCreator<T> helper,
                    ISimpleQueryListener<T> listener) {

    SimpleQueryBuilder<T> qb =
           new SimpleQueryBuilder<T>(helper, listener);

    AsyncContentResolver resolver = new
           AsyncContentResolver(context.getContentResolver());

    resolver.queryAsync(uri, null, null, null, null, qb);
  }
```

66

Simple Query

- Factory for queries

```
public class SimpleQueryBuilder<T>
                implements IContinue<Cursor> {

  @Override
  public void kontinue(Cursor cursor) {
    List<T> instances = new ArrayList<T>();
    if (cursor.moveToFirst()) {
       do {
          T instance = helper.create(cursor);
          instances.add(instance);
       } while (cursor.moveToNext());
    }
    cursor.close();
    listener.handleResults(instances);
  }
```

67

Simple Query

- Example client

```
public class BookstoreActivity extends Activity implements ... {

  @Override
  public void onCreate(Bundle savedInstanceState) {
    ...
    SimpleQueryBuilder.executeQuery(
        this,
        BookContract.CONTENT_URI,
        new IEntityCreator<Book>() { ... },
        this);
  }

  @Override
  public void handleResults(List<Book> books) {
    // update UI with book information
  }
```

68

Simple Query

- Example client

```
public class BookstoreActivity extends Activity {

  @Override
  public void onCreate(Bundle savedInstanceState) {
    ...
    SimpleQueryBuilder.executeQuery(
        this,
        BookContract.CONTENT_URI
        new IEntityCreator<Book>() { ... },
        new ISimpleQueryListener<Book>() {
           public void handleResults(List<Book> books) {
              ...
           }
        });
  }
```

69

Simple Query

- Factory for queries

```
public class SimpleQueryBuilder<T>
                implements IContinue<Cursor> {

  public static <T> void executeQuery(Activity context,
                    Uri uri,
                    String[] projection,
                    String selection,
                    String[] selectionArgs,
                    IEntityCreator<T> helper,
                    ISimpleQueryListener<T> listener) {

    ...

    resolver.queryAsync(uri, projection,
                        selection, selectionArgs, null, qb);
  }
```

70

Loader Queries

- Callback for loader-managed queries

```
public interface IQueryListener<T> {

   public void handleResults
               (TypedCursor<T> results);

   public void closeResults();
}
```

71

Loader Queries

```
public class QueryBuilder<T> implements
             LoaderManager.LoaderCallbacks<Cursor> {

  private QueryBuilder(String tag,
               Context context,
               Uri uri,
               int loaderID,
               IEntityCreator<T> creator,
               IQueryListener<T> listener) {
    ...
  }
```

72

Loader Queries

```
public class QueryBuilder<T> implements
    LoaderManager.LoaderCallbacks<Cursor> {

  public static <T> void executeQuery(String tag,
          Activity context, Uri uri,
          int loaderID,
          IEntityCreator<T> creator,
          IQueryListener<T> listener) {

    QueryBuilder<T> qb = new QueryBuilder<T>(tag, context,
                                uri, loaderID,
                                creator, listener);

    LoaderManager lm = context.getLoaderManager();

    lm.initLoader(loaderID, null, qb);
  }
```

73

Loader Queries

```
public class QueryBuilder<T> implements
    LoaderManager.LoaderCallbacks<Cursor> {

  @Override
  public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    if (id == loaderID) {
       return new CursorLoader(context,
                               uri,
                               projection,
                               select,
                               selectArgs,
                               null);
    }
}
```

74

Loader Queries

```
public class QueryBuilder<T> implements
    LoaderManager.LoaderCallbacks<Cursor> {

  @Override
  public void onLoadFinished(Loader<Cursor> loader,
                             Cursor cursor) {
    if (loader.getId() == loaderID) {
      listener
          .handleResults(new TypedCursor<T>(cursor, creator));
    } else {
       throw new IllegalStateException
                    ("Unexpected loader callback”);
    }
  }
```

75

Loader Queries

```
public class QueryBuilder<T> implements
    LoaderManager.LoaderCallbacks<Cursor> {

  @Override
  public void onLoaderReset(Loader<Cursor> loader) {
    if (loader.getId() == loaderID) {
      listener.closeResults();
    } else {
      throw new IllegalStateException
          ("Unexpected loader callback”);
    }
  }
```

76

Loader Queries

- Example client

```
public class BookstoreActivity extends Activity {

  @Override
  public void onCreate(Bundle savedInstanceState) {
    ...
    QueryBuilder.executeQuery(TAG,
        this,
        BookContract.CONTENT_URI,
        new IEntityCreator<Book>() { ... },
        new IQueryListener<Book>() {
           public void handleResults(TypedCursor<Book> books) {
              if (books.moveToFirst()) { ... }
           }
        });
  }
```

77

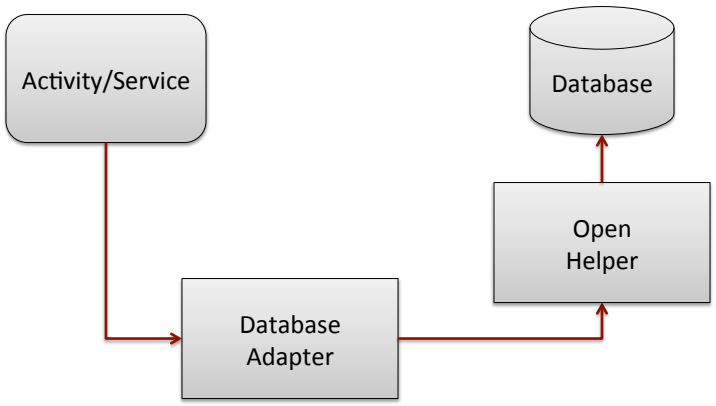
ENTITY MANAGER

78

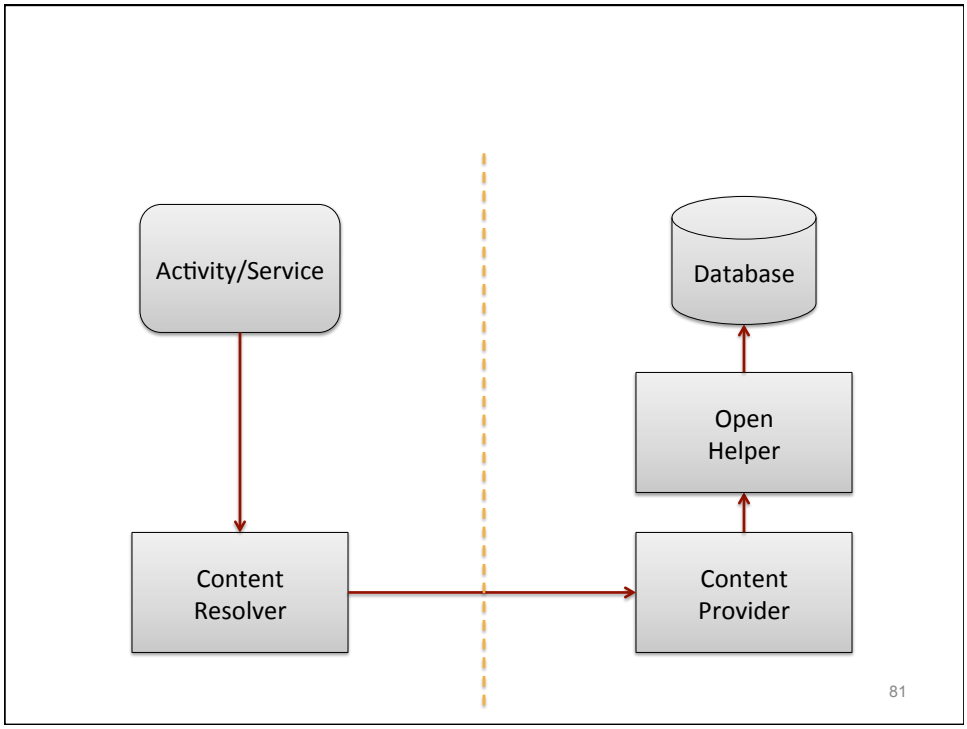
Entity Manager

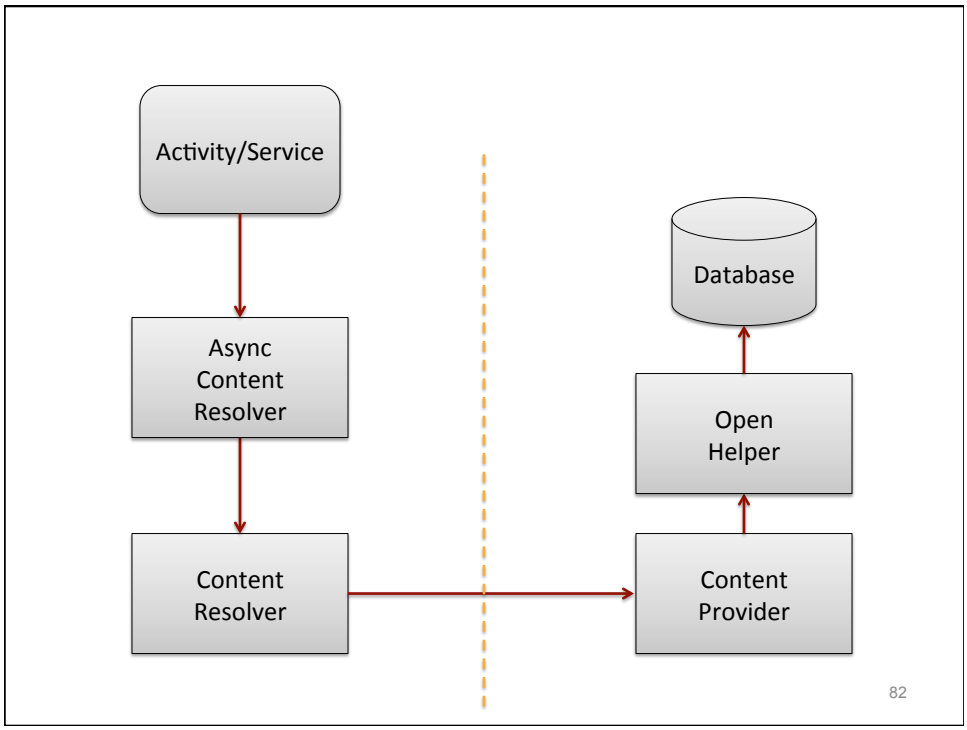
- Data Access Object (DAO) for entities
  - Client-side adapter for content provider
- Encapsulates asynchronous operations
- Encapsulates use of content provider
- Type-safe application-specific operations

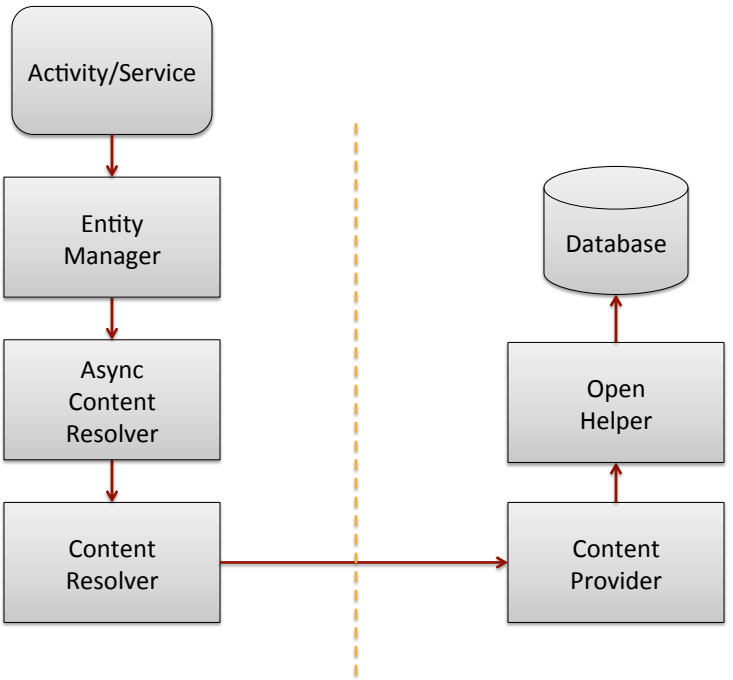
79



80


Activity/Service
Database
Open
Helper
Content
Resolver
Content
Provider
81


Activity/Service
Database
Async
Content
Resolver
Open
Helper
Content
Resolver
Content
Provider
82



83

Manager Class

```
public abstract class Manager<T> {

  private final Context context;

  private final IEntityCreator<T> creator;

  private final int loaderID;

  private final String tag;

  protected Manager(Context context,
                    IEntityCreator<T> creator,
                    int loaderID) {
    this.context = context;
    this.creator = creator;
    this.loaderID = loaderID;
    this.tag = this.getClass().getCanonicalName();
  }
```

84

Manager Class

```
public abstract class Manager<T> {

  private ContentResolver syncResolver;

  private AsyncContentResolver asyncResolver;

  protected ContentResolver getSyncResolver() {
    if (syncResolver == null)
      syncResolver = context.getContentResolver();
    return syncResolver;
  }

  protected AsyncContentResolver getAsyncResolver() {
    if (asyncResolver == null)
      asyncResolver =
        new AsyncContentResolver(context.getContentResolver());
    return asyncResolver;
  }
```

85

Manager Class

```
public abstract class Manager<T> {

  protected void executeSimpleQuery(Uri uri,
                 ISimpleQueryListener<T> listener) {
    SimpleQueryBuilder.executeQuery((Activity) context,
                               uri, creator, listener);
  }

  protected void executeSimpleQuery(Uri uri,
                 String[] projection,
                 String selection, String[] selectionArgs,
                 ISimpleQueryListener<T> listener) {
    SimpleQueryBuilder.executeQuery((Activity) context,
            uri, projection, selection, selectionArgs,
            creator, listener);
}
```

86

Manager Class

```
public abstract class Manager<T> {

  protected void executeQuery(Uri uri, IQueryListener<T> listener) {
    QueryBuilder.executeQuery(tag, (Activity) context, uri, loaderID,
                              creator, listener);
  }

  protected void executeQuery(Uri uri, String[] projection,
          String selection, String[] selectionArgs,
          IQueryListener<T> listener) {
    QueryBuilder.executeQuery(tag, (Activity) context, uri, loaderID,
          projection, selection, selectionArgs, creator, listener);
  }

  protected void reexecuteQuery(Uri uri, String[] projection,
          String selection, String[] selectionArgs,
          IQueryListener<T> listener) {
    QueryBuilder.reexecuteQuery(tag, (Activity) context, uri, loaderID,
          projection, selection, selectionArgs, creator, listener);
  }
```

87

Synchronous vs Asynchronous Operations

- Synchronous
  - Use on background threads

```
Uri insert(Book book);
```

- Asynchronous (UI thread)
  - (Typically) Continuation as last parameter
  - Name ends with `Async` (our convention)

```
void insertAsync(Book book,
                 IContinue<Uri> callback);
```

88

Synchronous vs Asynchronous Operations

- Synchronous
  - Use on background threads

```
Book search(String title);
```

- Asynchronous (UI thread)
  - (Typically) Continuation as last parameter
  - Name ends with `Async` (our convention)

```
void searchAsync(String title,
                 IContinue<Book> callback);
```

89

Synchronous vs Asynchronous Operations

- Synchronous
  - Use on background threads

```
List<Book> search(String title);
```

- Asynchronous (UI thread)
  - (Typically) Continuation as last parameter
  - Name ends with `Async` (our convention)

```
void searchAsync(String title,
          ISimpleQueryLister<Book> listener);
```

90

Synchronous vs Asynchronous Operations

- Synchronous
  - Use on background threads

```
TypedCursor<Book> getAll();
```

- Asynchronous (UI thread)
  - (Typically) Continuation as last parameter
  - Name ends with `Async` (our convention)

```
void getAllAsync(IQueryListener<Book>
                 listener);
```

91

FILE PROVIDER

92

File Content

- Remote access to files
- Server side:

```
public ParcelFileDescriptor openFile(Uri uri,
                                     String mode) {
   File file = ... // identify file from uri
   int imode = ParcelFileDescriptor.MODE_WRITE_ONLY;
   return ParcelFileDescriptor.open(file, imode);
}
```

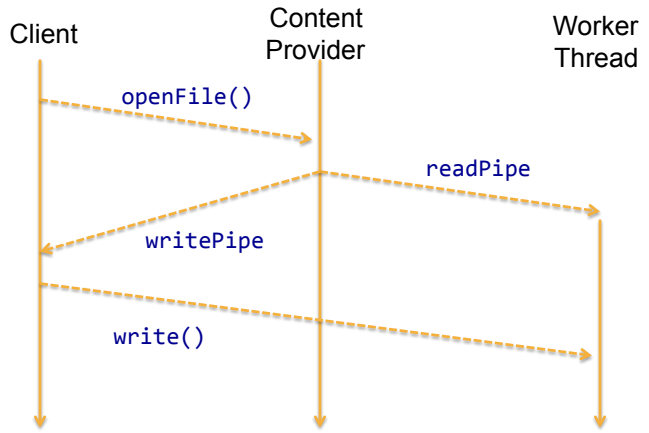
- Client side:

```
getContentResolver().openOutputStream(uri)
```

93

File Processing

- Example: Want to check integrity of new files



94

File Processing with Pipes

```
public ParcelFileDescriptor openFile(Uri uri, String mode) {
   ParcelFileDescriptor[] pipe =
      ParcelFileDescriptor.createPipe();
   final ParcelFileDescriptor readPipe = pipe[0];
   ParcelFileDescriptor writePipe = pipe[1];

   final File file = ... // identify file from uri
   new Thread(new Runnable() {
      public void run() {
         InputStream is = new AutocloseInputStream(readPipe);
         // read into a buffer and check integrity
         OutputStream os = new FileOutputStream(file);
         // copy from the buffer to this output file
      }
   }).start();

   return writePipe;
}
```

95

CONCLUSIONS

96

Content Providers vs Databases

- Pro:
  - Shares data between apps
  - Supported by cursor loader & loader manager
  - Cache database connections
  - All operations routed through singleton object
    - Synchronization not relying on database
- Con:
  - Bureaucracy
  - Security (`exported = "false"` in Manifest)
  - Encapsulate transactions

97

Resource Leaks

- Always close your cursor
  - *Unless* using a cursor loader & loader manager
- Always close your DB connections
  - *Except* in a content provider

98