Android: annotations to the rescue

Romain Rochegude

2016.03.30

Introduction

Introduction

- Huge gain of productivity
  - Don't (re)write boilerplate code
  - Code generation
- Code quality improvements
  - Widely tested and documented third-party libraries
  - Less code to write, less bugs to solve
- What about performances?
  - Annotations processing compile time vs. runtime

Android compliance

Android compliance

- android-apt plugin for Android Studio
  - Allows developers to configure a compile time annotation processor as a dependency in the `build.gradle` file
  - Runs annotation processing
  - Example:

```
dependencies {
    compile 'a.group:annotation:x.x.x'
    apt 'a.group:processor:x.x.x'
}
```

Views

Butter Knife

- No `findViewById` anymore, elegant binding mechanism
- Still want to use the `findViewById`? Just try the autocast alternative
- Simple way to bind resources
- Bind anything: activities, fragments, views, view holders
- The power of view lists (actions, setters)

- Event bindings:
  - OnClick/OnLongClick, OnItemSelected, OnCheckedChanged, OnEditorAction, OnFocusChange, OnItemLongClick, OnPageChange, OnTextChanged, OnTouch

- Under the hood
  - Same package binding class generation
  - A single entry point: the `ButterKnife` class, that resolves the concrete binder

TODO: insert code

Navigation

IntentBuilder and FragmentArgs

- Boilerplate code to declare a new screen
- Annotations to declare an Activity/Fragment
- Annotations to declare (optional) parameter(s) to pass
- Class generation following the Builder pattern
- Method to inject parameter(s) in the target class

TODO: insert code

Interactions

AutoValue: Cursor Extension

- Abstract class with the `@AutoValue` to define POJO
- Simple `@ColumnName` to define the binding between column names and class fields

```java
@AutoValue
public abstract class Person {
 @ColumnName("name") abstract String name();

 public static Person create(Cursor cursor) {
  return AutoValue_Person.
          createFromCursor(cursor);
 }

 abstract ContentValues toContentValues();
}
```

TODO: insert generated code

Others

Others

- ORM: requery
- JSON parsing: LoganSquare
- Bus: EventBus
- Saving and restoring instance state: Icepick
- Easily deal with the result from an activity started for result: OnActivityResult
- and so on: `http://android-arsenal.com/tag/166`

Write custom annotation processors

Concepts

- Provide a robust annotation API
- Implement the algorithm to search for your annotations and deal with it

Generate Java source files: JavaPoet

- Powerful and complete API to describe
  - static imports,
  - classes, interfaces, enums, anonymous inner classes,
  - fields, parameters, variables,
  - methods, constructors,
  - annotations, javadoc
- Specific wildcards to format the output code
- Test generated files with *TODO*

```
MethodSpec main = MethodSpec
    .methodBuilder("main")
    .addModifiers(Modifier.PUBLIC,
                  Modifier.STATIC)
    .returns(void.class)
    .addParameter(String[].class, "args")
    .addStatement("$T.out.println($S)",
                  System.class,
                  "Hello, JavaPoet!")
    .build();
```

to

```
public static void main(String[] args) {
    System.out.println("Hello, JavaPoet!");
}
```

Google AutoService

- Simple `@AutoService` annotation for your `javax.annotation.processing.Processor` subclass
- Automatic generation of the `META-INF/services/javax.annotation.processing.Processor` in the output classes folder
- Automatic inclusion of the annotated processor

A must-read article: ANNOTATION PROCESSING 101 by Hannes Dorfmann

Conclusion

Conclusion

- Performance: machine and/or human
- Readable
- Maintainable