Android – Eine Einführung

Testen & Veröffentlichen

Andreas Wilhelm

26. Juli 2014

Testen von Applikationen

Contents

Allgemeines

- Android-SDK bringt Emulator und Debugging-Monitor *LogCat* mit
- Integration von JUnit unter Android
- *monkeyrunner*, eine auf Python basierende Stress-Test-API
- JUnit-Tests sind Emulator basiert
- Mock-Framework zwar theoretisch vorhanden praktisch unbrauchbar $\rightarrow$ Alternativ-Projekte interessant

Allgemeines

- Eigenes Android-Projekt mit Test-Paketen und -Klassen
- Eine Klasse implementiert die Tests eines Moduls
- Eine Klassen-Methode einen Test
- Android lädt Test- und das Applikations-Paket mit TestRunner

Test-Projekt Erstellen

- Einbindung der SDK-Test-Tools über Eclipse ADT-Plugin
- Erstellen des Projekt-, Quellcode- und Ressourcen-Verzeichnisses
- Anlegen eines Test-Pakets mit Suffix *.test*
- Generieren des Android-Manifests und der ANT-Dateien
- Eintragung des *InstrumentationTestRunner* im Manifest
- Platzieren der Test-Klassen im Source-Ordner (*src/*)

JUnit Basis

- Java-Tests ohne Android-API-Zugriffe mit *TestCase*
- Andernfalls Verwendung von *AndroidTestCase*
- Spezialisierte Klassen, wie *ApplicationTestCase*, *LoaderTestCase*, *ProviderTestCase2* oder *ServiceTestCase*
- Alle Klassen ermöglichen Verwendung von JUnit-Assertions

Instrumentation

- Einflussnahme auf Lebenszyklus einer Applikation wichtig
- Android Klasse *Instrumentation*
- Simulieren das beispielsweise Starten (*onCreate()*) oder Beenden (*onDestroy*) einer Activity
- Beispielanwendung: Prüfung des gespeicherten Zustands einer Activity
- Spezialisierte Klassen, wie *ActivityInstrumentationTestCase2*

AndroidTestCase

AndroidTestCase bringt weitaus mehr als Methoden *setUp()* und *tearDown()* mit. Sie enthält Methoden, die das Testen von Zugriffsrechten ermöglichen und eine Methode, die Speicher-Lecks verhindert, indem sie Klassen-Referenzen entfernt.

Beispiel

```
public class UbuntuFeedbackTest extends
    ActivityInstrumentationTestCase2<Feedback> {
    public static final String FEEDBACK_NAME_INITIAL = "";
    public static final String FEEDBACK_NAME_DESTROY = "Stephe Ericson";

    private Feedback feedbackActivity;
    private TextView feedbackName;

    public UbuntuFeedbackTest() {
        super(Feedback.class);
    }

    public UbuntuFeedbackTest(Class<Feedback> activityClass) {
        super(activityClass);
    }

    @Override
    protected void setUp() throws Exception {
        // Setup the test case, ...
        super.setUp();

        // ... disable the touch mode ...
        setActivityInitialTouchMode(false);

        // ... and fetch the forced activity.
        this.feedbackActivity = getActivity();
    }

    public void testInstanceState() {
        // Fetch the field for the sender name ...
        this.feedbackName = (TextView) this.feedbackActivity.findViewById(net.avedo.ubuntu.releases.R.id.
                txtFeedbackName);

        // ... and check the initial state.
        assertEquals(FEEDBACK_NAME_INITIAL, feedbackName.getText().toString());

```

Anmerkungen

- Test-Paket und Activity werden nicht im selben Thread geladen
- Zugriff auf grafische Oberfläche mit *runOnUiThread()*
- Alternative: Annotation *@UiThreadTest*

Beispiel

```
@UiThreadTest
public void testInstanceState2() {
    // Fetch the field for the sender name ...
    this.feedbackName = (TextView) this.feedbackActivity.findViewById(net.avedo.ubuntu.releases.R.id.
            txtFeedbackName);

    // ... and check the initial state.
    assertEquals(FEEDBACK_NAME_INITIAL, this.feedbackName.getText().toString());

    // Change the text, ...
    this.feedbackName.setText(FEEDBACK_NAME_DESTROY);

    // ... stop the activity (call onDestroy method) ...
    this.feedbackActivity.finish();

    // ... and restart it (call onResume method).
    this.feedbackActivity = getActivity();

    // Fetch the field for the sender name ...
    TextView nameAfterRestart = (TextView) this.feedbackActivity.findViewById(net.avedo.ubuntu.releases.R.id.
            txtFeedbackName);

    // ... and check the initial state.
    assertEquals(FEEDBACK_NAME_DESTROY, nameAfterRestart.getText().toString());
}
```

Listing : Die *@UiThreadTest* Annotation

Anmerkungen

- Bibliotheken für das Testen von Activities, Services und ContentProvider
- Vor- und Nachbereitung der Tests (*setUp()* und *tearDown()*)
- Mock-Objekte als Platzhalter für echte Objekte

BroadcastReceiver

Android stellt keine Bibliothek für das Testen von BroadcastReceivern bereit. Daher muss die Komponente getestet werden, die das Intent an den Receiver sendet. Dabei überprüft man, ob der Receiver korrekt antwortet bzw. reagiert.

Applikation testen

- *ApplicationTestCase* testet Applikation ansich
- Starten und Beenden der Applikation
- Kein Zugriff auf einzelne Komponenten
- Validierung der Angaben in Manifest

Assertions

- Android SDK bietet aus JUnit bekannte Assertions
- Komplexere Überprüfungen durch *MoreAsserts*
- View bezogene Assertions durch *ViewAsserts*
  $\rightarrow$ Dient Prüfung von Bemaßung und Positionierung

Mock-Objekte

- Minimierung der Abhängigkeiten durch *Dependency Injection*
- Verschiedene Komponenten, wie *Context*-, *ContentProvider*- oder *Service*-Objekte
- Teilweise sogar Nachbildung durch Mock-Intents
- Leider nur überschriebene Methoden
  → Werfen *UnsupportedOperationException*
  → Ableitung der Klassen nötig
- Interessant *MockContentResolver* sind vorerst keinem Provider zugeordnet
  → Explizite Zuweisung mit *ContentResolver.add(String, ContentProvider)*

MockApplication	MockContext	MockContentProvider
MockContentResolver	MockPackageManager	MockResources
MockCursor	MockDialogInterface	

Tabelle: Mock-Objekte

Mock-Context

- Nachbildung globaler Schnittstellen
- Zwei Context-Mock-Objekte

IsolatedContext Klasse stellt einen isolierten Kontext zur Verfügung, der Operationen auf Dateien, Datenbanken oder Verzeichnissen in Testumgebung ausführt, was den Funktionsumfang einschränkt.

RenamingDelegatingContext Stellt einen Kontext zur Verfügung, der fast alle Funktionen durch ein normales *Context*-Objekt ausführen lässt und nur Datei- und Datenbank-Operationen in einem *IsolatedContext* ausführt.

Allgemeines

- Ausführung von Tests durch *Test-Runner-Klasse*
- Lädt Tests und zu testendes Projekt
- Vorbereiten (*setUp()*), Ausführen (*run()*) und Nachbereiten (*tearDown()*) jedes einzelnen Tests
- Standard-Runner-Klasse *InstrumentationTestRunner* (erweitert JUnit-Test-Runner)
- Deklaration des Test-Runners im Manifest über *<instrumentation>*-Element
- Laden einer Bibliothek im Manifest mit *<uses-library>*-Element

Das Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.avedo.ubuntu.releases.test"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <instrumentation
        android:name="android.test.InstrumentationTestRunner"
        android:targetPackage="net.avedo.ubuntu.releases" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:allowBackup="false" >
        <uses-library android:name="android.test.runner" />
        <activity android:name="net.avedo.ubuntu.releases.test.UbuntuReleaseTest" />
    </application>
</manifest>
```

Listing : Ein Manifest für Test-Projekte

Fazit

- Unterscheidung zwischen JUnit und Android-JUnit
- Zusätzliche Einbindung von Android-Assertions
- Test-Klassen für Komponenten
- Verschiedene Mock-Objekte
- Spezieller Test-Runner

Probleme

- Android-JUnit basiert auf JUnit 3 (nicht auf 4)
- Tests werden im Emulator ausgeführt
- Applikation wird für jeden Test neu gepackt und gestartet
  $\rightarrow$ Sehr lange Test-Zeiten
- Manche Dinge nur sehr schwer zu testen (Adapter & Menüs)
- Unzureichende Mock-Objekte
- Dalvik VM unterstützt keine Java Reflections (benötigt von bekannten Mocking Frameworks)
- *android.jar* enthält nur unvolständige Class-Dateien
  $\rightarrow$ Android-Spezifische Klassen werfen auserhalb der Dalvik VM *RuntimeException*

Alternativen

- Bekannte Mocking-Frameworks, wie *Mockito*, *EasyMock* oder *jMock*
  → Benötigen Java Reflections
- Android-Wrapper für EasyMock (*android-Mock*)
  → Erzeugt die Mock-Objekte bereits zur Compile-Zeit
- *Robolectric* ermöglicht Erzeugen von Android-Komponenten direkt in JVM (ohne *RuntimeException*)
  → Normale Verwendung von *Mockito* oder *EasyMock*
  → Erlaubt das schreiben normaler JUnit-Tests

Robolectric-Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.avedo.ubuntu.releases.test"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <instrumentation
        android:name="android.test.InstrumentationTestRunner"
        android:targetPackage="net.avedo.ubuntu.releases" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:allowBackup="false" >
        <uses-library android:name="android.test.runner" />
        <activity android:name="net.avedo.ubuntu.releases.test.UbuntuReleaseTest" />
    </application>
</manifest>
```

Listing : Ein Manifest für Test-Projekte

Google Play

Contents

Allgemeines

- *Google Play Store* ist der offizielle Android-Markt
  → Ausnahme *Kindle Fire* HD
  → Erfordert *Publisher Account*
  → Benötigt *Google Checkout Merchant Account* für Verkäufe
- Vertrieb von Applikationen, Musik, Büchern und Filmen
- Alternativ privater Vertrieb (Homepage)
- Amazon-Market

Registrierung

- *Google Play Store* erfordert *Publisher Account*
  $\rightarrow$ Einmalig 25$
  $\rightarrow$ https://play.google.com/apps/publish/
- Abfrage von Name, Passwort, Geburtsdatum, usw.
- Anlegen eines Entwickler-Profils (Name, Website und Telefonnummer)
- *Vereinbarung für den Entwicklervertrieb* (landesspezifisch)
- Überweisung der 25$ mit *Google Checkout*

Checkliste

- Prüfen ob der Paketname aussagekräftig ist (kann nicht mehr geändert werden)
- Entfernen des *android:debuggable*-Attibuts aus dem Android-Manifest
- Entfernen aller Debug- und Log-Ausgaben
- Sicherstellen, dass keine Test-Ressourcen (beispielsweise Test-Server) mehr verwendet werden
- Verzeichnis-Struktur des Projekts bereinigen – Jedes Verzeichnis sollte nur die für es vorgesehenen Dateien enthalten
- Entfernen unnötiger RAW-Dateien aus *assets/* und *res/raw/*
- Überprüfung der gesetzten Rechte in Android-Manifest
- Hinterlegen eines Namens und eines Icons in Android-Manifest
- Android-SDK-Versionen in Android-Manifest überprüfen
- *android:versionCode* und *android:versionName* in Android-Manifest prüfen

Release-APK

- Signieren mit privatem Schlüssel
  → Erzeugung mit *Keytool*
  → Signieren mit *Jarsigner*
- Überarbeitung des Pakets mit *zipalign* (SDK-Werkzeuge)
- Hochladen des *.apks über *Publisher Konsole*
- Einstellung von Vertriebsländern, Preisen
- Erstellung von Dokumentation und Screenshots
- Kategorisierung
- Veröffentlichen

Google-Prüfung

Seit Kurzem werden alle neu in den Play Store hochgeladenen Pakete vor dem Vertrieb durch ein automatisiertes Google-Tool geprüft um Schadprogramme zu filtern.