Android - Postpone start, release, libraries, practical know how

Lukas Prokop

24th November 2017

Postpone starting

- Timer
- Handler
- AlarmManager
- JobScheduler
- GCMNetworkManager

Timer and TimerTask

- `Timer` allows to run `TimerTask` in defined time or repeatedly
- Creates new thread where it runs
  - One thread per timer
- For updating UI needs to call `runOnUIThread()`
- Not recommended to use -> Use `Handler` instead
- Timer can schedule multiple TimerTask
- TimerTask is not reusable

Timer and TimerTask

```
long delay = 10000L;
long period = 10000L;
Timer timer = new Timer();

TimerTask myTimerTask = new TimerTask() {
   @Override
   public void run() {
       doSomeStuff();
   }
};

timer.schedule(myTimerTask, delay);
timer.schedule(myTimerTask, new Date(System.currentTimeMillis() + delay));
timer.schedule(myTimerTask, delay, period);
timer.schedule(myTimerTask, new Date(System.currentTimeMillis() + delay),
 period); // Fixed delay between subsequent tasks

timer.scheduleAtFixedRate(myTimerTask, delay, period);
timer.scheduleAtFixedRate(myTimerTask, new Date(System.currentTimeMillis() +
 delay), period); // runs tasks exactly after the period if it is posssible
```

Handlers

- Possible to run on background or UI thread
- Possible for scheduling or delaying start of some “task”
- In case of device sleep handler doesn’t run
- Messages
  - `sendMessageAtTime(Message msg, long uptimeMillis)`
  - `sendMessageDelayed(Message msg, long delayMillis)`
- Runnable
  - `postAtTime(Runnable r, long uptimeMillis)`
  - `postDelayed(Runnable r, long delayMillis)`
- Good for task with high frequency (more than one in few minutes)
- Tight with application component

Handler - repeat

```
Handler mHandler = new Handler();

private void handlerRepeat() {
   Runnable r = new Runnable() {
       @Override
       public void run() {
           updateUI();
           mHandler.postDelayed(this, 5000L);
       }
   };

   mHandler.postDelayed(r, 5000L);
}
```

Alarm manager

- Perform time-based operations outside the lifecycle of application
- Fire intents at specified time
- In conjunction with broadcast receivers start services
- Operate outside of your application, trigger events or actions even app is not running or device is asleep
- Minimize app resource requirements
- Action is specified by `PendingIntent`
- Evolved a lot in time
  - Added some new method
  - Some method changed behaviour from exact -> inexact
  - READ the documentation carefully

Alarm manager - tips

- For synchronization consider to use GCM together with SyncAdapter
- For repeating sync add some randomness when it is syncing
  - Imagine 1M+ of devices trying to download something from your server at the same time
- Use `setInexactRepeating` if it is possible to group alarms from multiple apps => Reduces battery drain
- Alarms are cancelled on reboot, reschedule alarms when device boots

Alarm manager - alarm type

- `ELAPSED_REALTIME`
- `ELAPSED_REALTIME_WAKEUP`
- `RTC`
- `RTC_WAKEUP`

- Clock types
  - Elapsed - time since system boot
    - Use when there is no dependency on timezone
  - Real time clock - time since epoch
    - Use when you need to consider timezone/locale
- Wake up
  - wakeup - ensure alarm will fire at the scheduled time
  - non wakeup - alarm are fired when device awakes

AlarmManager - important changes

- API < 19 (KITKAT) - `set*` methods behave like exact time
- API > 19
  - All old methods are inexact now
  - New API for setting exact alarm
    - setExact
  - Added new API for specify windows, when it should be delivered
    - setWindow
- API 21
  - Added methods `setAlarmClock` and `getNextAlarmClock`
  - system can show information about alarm
- API 23
  - Added methods `setExactAndAllowWhileIdle` and `setAndAllowWhileIdle`
- API 24
  - Added direct callback versions of `set` and `setExact` and `setWindow`

AlarmManager - usage

```
AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);

Intent i = new Intent("AlarmAction");
PendingIntent alarmIntent = PendingIntent.getBroadcast(getApplicationContext(),
 ALARM_REQUEST_CODE, i, PendingIntent.FLAG_UPDATE_CURRENT);

am.set(AlarmManager.RTC, System.currentTimeMillis() + TimeUnit.HOURS.toMillis(1L),
 alarmIntent);
```

- AlarmType
- Time
  - Depending on the alarm type it is timestamp or time since device boots
- PendingIntent
  - PendingIntent which specify action which should happen

Alarm manager - sleeping device

- Alarm manager can wake devices, when it asleep BUT
- pending intent is able to start activity/service or send broadcast
- BUT it is not guaranteed by system to start service/activity before device fall asleep again
- only `BroadcastReceiver.onReceive` is guaranteed to keep device awake
  - If you start activity/service in receiver, there is no guarantee that activity/service will start before the wake lock is released

Wake locks

- Prevent device from sleep
- Requires permission `android.permission.WAKE_LOCK`
- Multiple levels
  - `PARTIAL_WAKE_LOCK`
    - CPU is running, screen and keyboard backlight allowed to go off
  - `FULL_WAKE_LOCK`
    - Screen and keyboard on full brightness
    - Released when user press power button
  - `SCREEN_DIM_WAKE_LOCK`
    - Screen is on, but can be dimmed, keyboard backlight allowed to go off
    - Released when user press power button
  - `SCREEN_BRIGHT_WAKE_LOCK`
    - Screen on full brightness, keyboard backlight allowed to go off
    - Released when user press power button

Alarm manager - sleeping device, solution

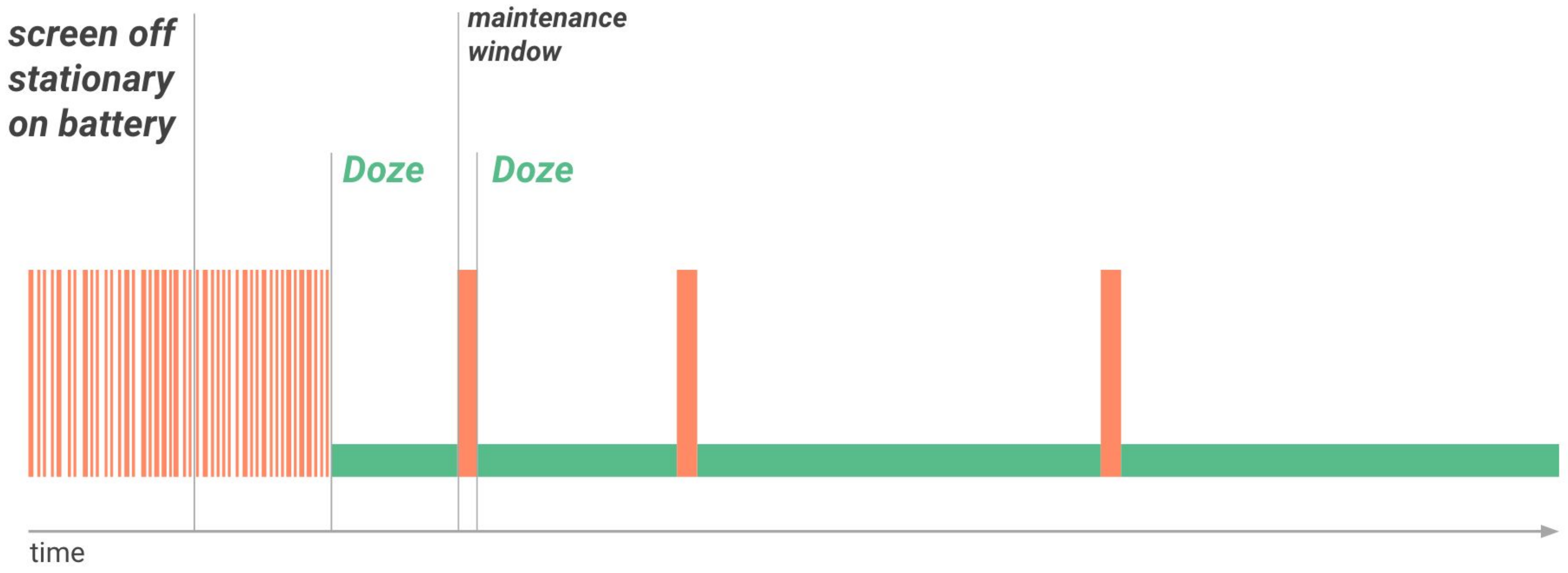
- Acquire your wake lock during `BroadcastReciver.onReceive` and before starting service
- Start service
- When service finish its job release the wake lock
  - It is really important to release wake lock, it disables turning off CPU

WakefulBroadcastReceiver

- Part of v4 support library

```
public class SimpleWakefulReceiver extends WakefulBroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        // This is the Intent to deliver to our service.
        Intent service = new Intent(context, SimpleWakefulService.class);

        // Start the service, keeping the device awake while it is launching.
        startWakefulService(context, service);
    }
}

public class SimpleWakefulService extends IntentService {
    public SimpleWakefulService() {
        super("SimpleWakefulService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        for (int i=0; i<5; i++) {
            Log.i("SimpleWakefulReceiver", "Running service " + (i+1)
                    + "/5 @ " + SystemClock.elapsedRealtime());
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
            }
        }
        SimpleWakefulReceiver.completeWakefulIntent(intent);
    }
}
```

Doze mode


screen off
stationary
on battery
maintenance
window
Doze
Doze
time

Doze mode

- Since API 21 (Lollipop
- Restrict app access to network and cpu intensive services
- Defers jobs, sync and alarms

Doze mode

- Network access is suspended.
- The system ignores wake locks.
- Standard AlarmManager alarms (including `setExact()` and `setWindow()`) are deferred to the next maintenance window.
  - If you need to set alarms that fire while in Doze, use `setAndAllowWhileIdle()` or `setExactAndAllowWhileIdle()`.
  - Alarms set with `setAlarmClock()` continue to fire normally — the system exits Doze shortly before those alarms fire.
- The system does not perform Wi-Fi scans.
- The system does not allow sync adapters to run.
- The system does not allow JobScheduler to run.

Job Scheduler

- Not for exact time schedule
- Possible to specify connectivity, charging, idle conditions
- System batch “jobs”
- Since API 21
- Battery efficient
- Job parameters defined in `JobInfo`
  - Backoff policy
  - Periodic
  - Delay triggers
  - Deadline
  - Persistency
  - Network type
  - Charging
  - Idle

Job Scheduler

```
JobScheduler jobScheduler
          = (JobScheduler) getSystemService(JOB_SCHEDULER_SERVICE);
ComponentName componentName = new ComponentName(this, MyJob.class);
jobScheduler.schedule(new Builder(1, componentName)
   .setBackoffCriteria(TimeUnit.HOURS.toMillis(5),
          JobInfo.BACKOFF_POLICY_EXPONENTIAL)
   .setPersisted(true)
   .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
   .setRequiresCharging(false)
   .build());
```

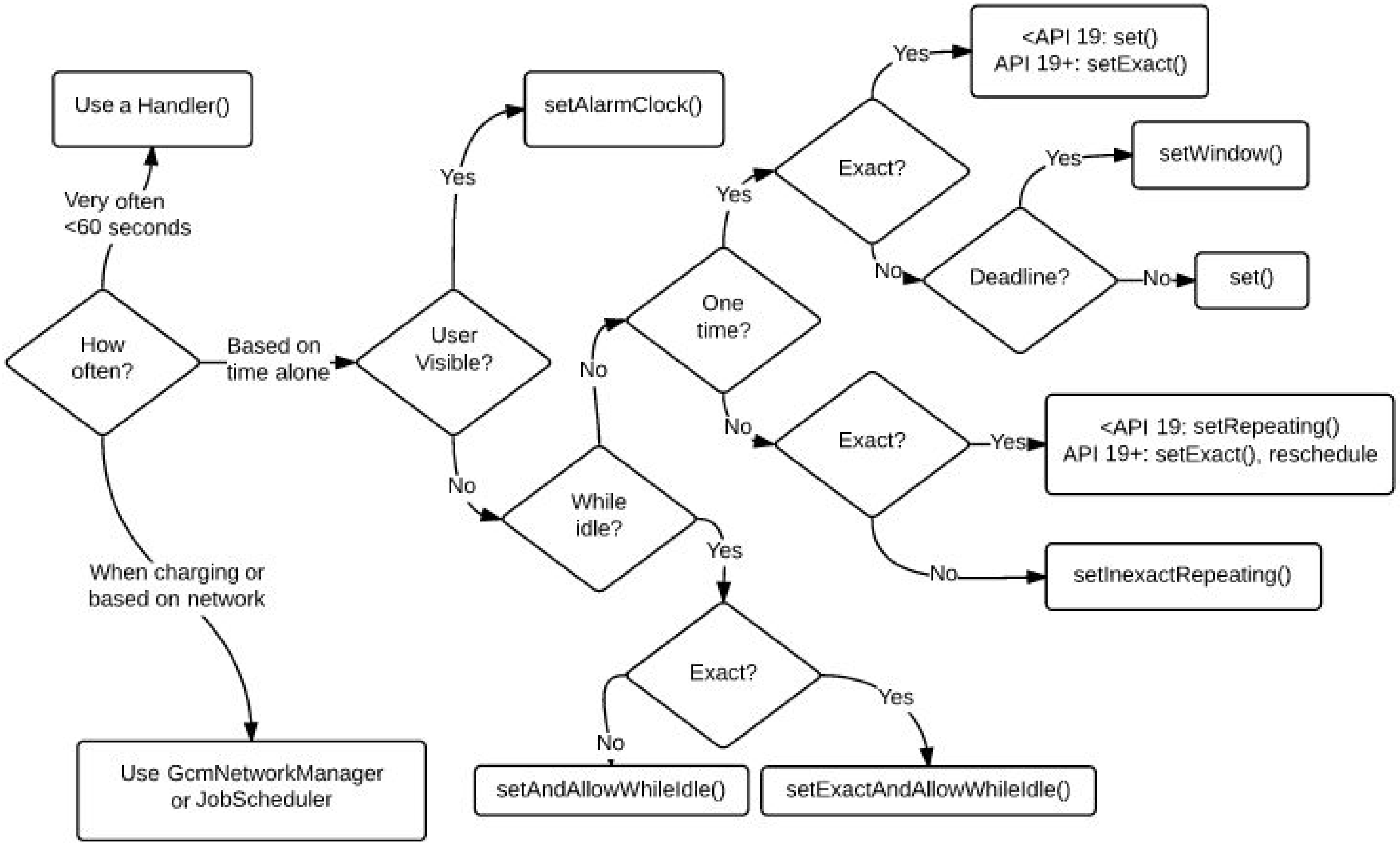
JobScheduler

```java
public class MyJob extends JobService {
    @Override
    public boolean onStartJob(JobParameters params) {
        // Do the job
        jobFinished(params, false);

        return false; // no more work to do with this job service
    }

    @Override
    public boolean onStopJob(JobParameters params) {
        // Called by system, before #jobFinished was called
        return false; // false to drop the job
    }
}
```

Firebase JobDispatcher

- Part of firebase
- Similar functionality and API as JobScheduler
- Uses JobScheduler on API > 21

How to decide what to use



OR

Android-job library

http://evernote.github.io/android-job/

Release process

- Gather materials for release
- Configure application for release
- Build application for release
- Prepare remote server
- Test your application

Gather materials for release

- Cryptographic keys
  - Apps in store are digitally signed, by developer certificate
  - Identify app developer
  - Self-signed certificate is enough
  - Needs to end after 22 October 2033
  - Protect your private key and password, it is not possible to update application if it is lost
- Application icon
  - Visible in launcher, settings or other applications
  - High-res assets for google play store listing
- End-user license agreement (EULA)
  - User should know if you gather some data
  - Not required but recommended
  - GDPR
- Misc. materials
  - Promo and marketing materials
  - Promotional text and graphic for store listing

Configure application for release

- Delete unused parts (sources, resources, assets)
- Package name
  - https://play.google.com/store/apps/details?id=com.avast.android.mobilesecurity
- Turn off debug features
  - Delete Log.* calls
  - Remove android:debuggable flag from AndroidManifest
  - Remove Debug or Trace calls
  - If you using WebView ensure that debug is disabled, otherwise is possible to inject js code
- Clean up your directories, checks if libraries doesn't include some unnecessary files to your *.apk (*.proto, java manifests, …)
- Review
  - permissions - remove unnecessary
  - App icon and label
  - Version code and version name
  - Used URLs test vs. production backend
- Check compatibility
  - Support of multiple screens
  - Tablet mode

Build application for release

- Signing
  - Manually
    - Using Keytool and Jarsigner from JDK
  - Configure sign options in gradle
  - Android studio
  - Sign scheme v2 https://source.android.com/security/apksigning/v2

- Obfuscation
  - Use proguard to obfuscate your code, it is really easy to decompile application and find how it works, endpoints

- Consider
  - Who has access to your sign key
  - Signing server

Prepare external servers and resources

- Ensure that backend is running
- Check if app is switched to prod environment

Testing your application for release

- Regression test
- Test new features
- If it is possible test it on multiple devices, android versions
- Test multiple languages
- Check if it is looks good with RTL languages
- Check lint, if it doesn’t contain some several issues

Publishing

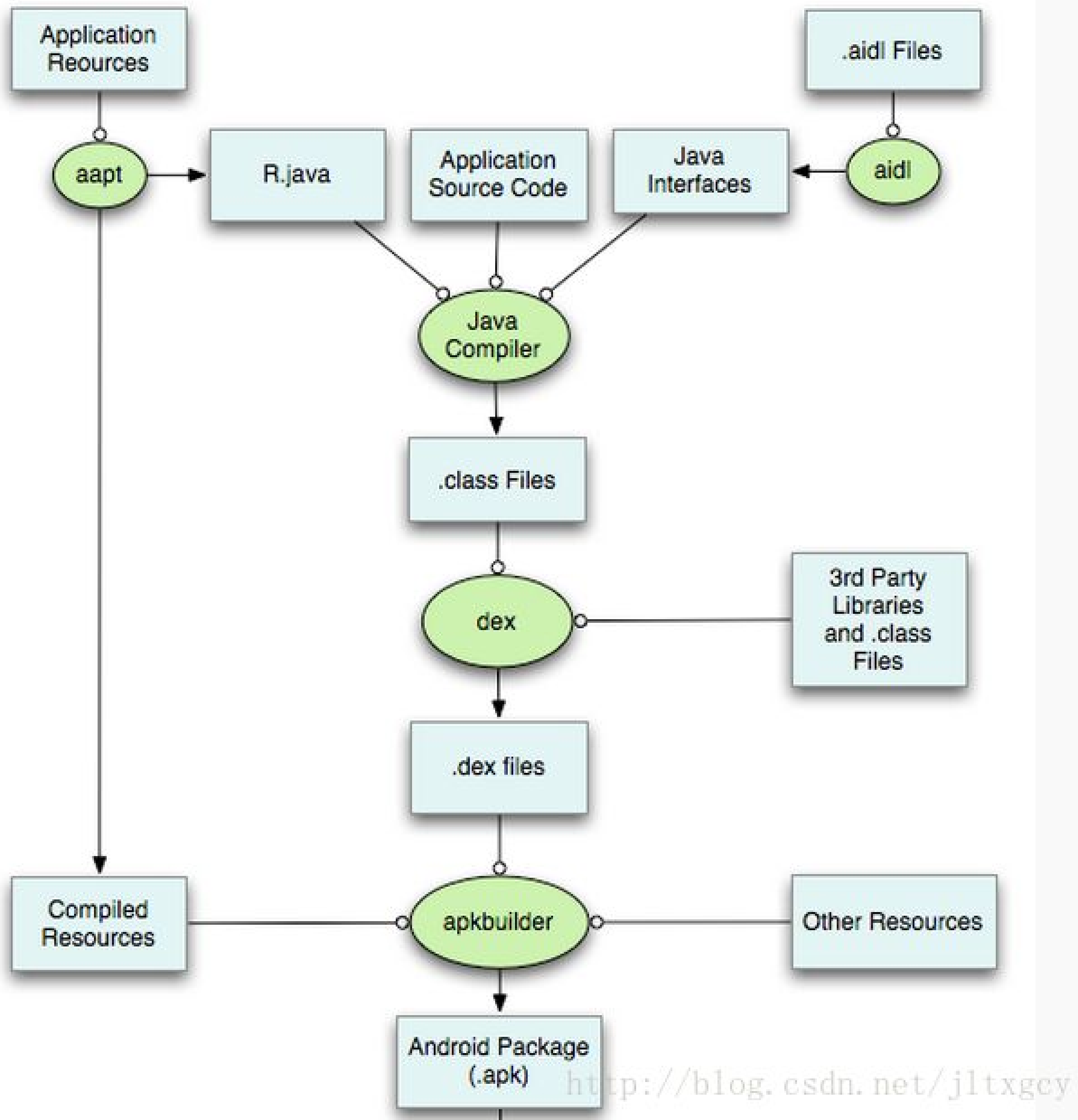
- Google play store
  - Registration cost 25 USD
  - Reporting about installs
  - Crashes
    - If user sends it
  - Cloud test lab
    - Run monkey tests on multiple devices before releasing
  - Alpha/Beta groups
  - Distribution specific domain - internal apps
  - Staged rollout
  - API - import crash reports to bug tracker, upload new APK, ...
- Email, Web page
  - Untrusted sources - security risk
  - Manual Updates
- 3rd party distribution
  - Amazon
  - Crashlytics/fabric - enables to update apps, crash reporting
    - Acquired by google, it should be part of the firebase soon

After publish

- Monitor new crashes
- New permission slows down spreading between users
- Not good idea publish app before weekend or vacation

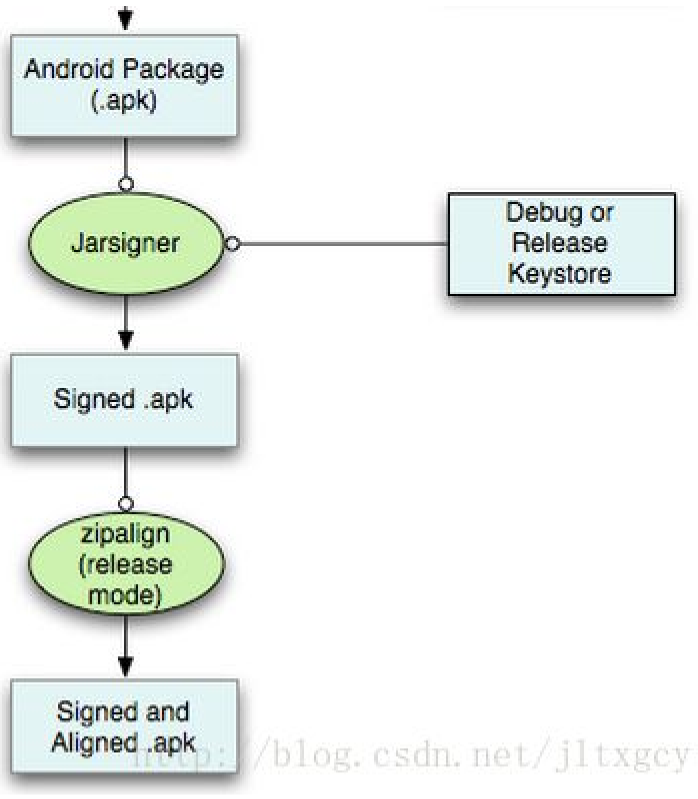
Build process and APK structure

Build process



avast

Build process



Building android apps limitations

- Dex limit 64k methods
  - Shrink unused methods and classes (libraries)
    - You need to specify what is entry point, build dependency graph. Classes which are not part of graph are removed, unused methods as well
    - When you work on library, provide proguard rules together with library
  - Multidex
    - Native support since API-21 for older version support library
    - Try to avoid using multidex, it slows down application start
    - Splits classes to multiple dex files
    - on API>=21 dex files are converted to single .oat file (ART runtime)
    - Main dex file loaded when app is started
    - Loading of additional dex files is performed during initialization
    - Dex files are in app folder
    - https://www.blackhat.com/docs/ldn-15/materials/london-15-Welton-Abusing-Android-Apps-And-Gaining-Remote-Code-Execution.pdf
  - Sometimes is better copy some classes from library into application
    - for example guava library
- Google doesn't allow code side load

APK structure

- Zip contains

```
.
├── AndroidManifest.xml- binary XML form of Android manifest
├── classes.dex        - classes compiled to dex
├── META-INF
│   ├── CERT.RSA        - Application certificate
│   ├── CERT.SF         - SHA-1 digest of corresponding lines in the 
MANIFEST.mf
│   └── MANIFEST.MF     - Manifest file list of files with their SHA-1 hash
├── assets              - assets files
├── res                - resources
│   ├── drawable-hdpi-v4
│   ├── drawable-mdpi-v4
│   ├── drawable-xhdpi-v4
│   ├── drawable-xxhdpi-v4
│   ├── drawable-xxxhdpi-v4
│   ├── layout
│   ├── menu
│   └── xml
└── resources.arsc           - resources compiled to binary form
```

Proguard

- Shrink - smaller code, faster build
- Optimize - faster code, removing static conditions
- Obfuscate - make it harder to read after decompilation

Decompile apk

- Unzip the apk
- Dex2Jar to convert classes.dex to jar archive
  - https://github.com/pxb1988/dex2jar
- jd-gui to view decompiled classes
  - http://jd.benow.ca/
- Android studio apk analyzer
  - Easy to check resources
  - Compare multiple apk

Android and SW development - best practices

- Keep strings, dimensions, colors, … in resources
- Create libs for parts used in multiple projects (simplify maintenance, speed-up builds)
- Use git
- Do code reviews
- Write tests

Libraries, gradle plugins, etc...

DexCount plugin

- Computes methods count in APK
- Visualize count in nice chart
- https://github.com/KeepSafe/dexcount-gradle-plugin

Butter Knife

- Field and method binding for android views
- Code is more readable
- Uses annotations for binding
- Generate code => fast
- http://jakewharton.github.io/butterknife/

Retrofit

- A type-safe HTTP client for Android and Java
- Simplify communication with some API service
- Configurable
  - OkHTTP 3 client - compression, timeouts
  - Supports multiple convertors
    - Gson
    - Jackson
    - Protobuf
    - Wire
- https://square.github.io/retrofit/

OkHttp

- An HTTP & HTTP/2 client for Android and Java applications
- Supports sync/async calls
- Supports multiple addresses per URL (Load balancing, failover)
- http://square.github.io/okhttp/

Dagger

- Dependency injection framework
- Decouple code
- Better testing
- https://google.github.io/dagger/android.html

Gson

- A Java serialization/deserialization library that can convert Java Objects into JSON and back.
- Customizable
- Possible to plug custom deserializer for specific object
- https://github.com/google/gson

Stetho

- Debug tool by Facebook
- Inspect
    - ViewHiearchy
    - Database
    - Shared preferences
    - Network trafic
- http://facebook.github.io/stetho

LeakCannary

- Helps with finding memory leaks
- https://github.com/square/leakcanary