Places, Bitmaps & more

Joe Rogers
Android 310

The More

Quick note on Background fetches

- One way to avoid overly fetching data for user is back off fetching based on the user.
  - Fetch data at fastest reasonable frequency the day after user opens the app.
  - As each day passes without use, reduce the frequency of background fetches.
  - Eventually the app should stop fetches if user doesn't open app. However, need to balance with user experience if first fetch would be bad.

Other background options

- Monitoring for broadcasts related to the user plugging in the device, battery level, and type of mobile network, etc
  - Allows you to vary based on situation. Plugged in eliminates battery consumption, but may still not want to fetch if on slow mobile network.

New support library yesterday

- Version 22.1 which is a huge release.
  - ActionBarActivity now called AppCompatActivity
    - Fully compatibile so just a matter of switching.
  - Use android:theme instead of app:theme for Toolbars.
    - Advantage is theming should work better for items in the toolbar. Ie spinners, custom text, etc.
  - On V11+ all views should support theming as well.
  - See resources for more details.

New testing library also

- Version 0.2 of testing library was also released.
  - Testing is highly evolving and anything I have showed you may not be the best way going forward.
  - Key item. New TestRules replacing the legacy classes. First classes are:
    - ActivityTestRule
    - ServiceTestRule
    - No ProviderTestRule yet, but suspect its coming.

Google Play Services

What it is

- Google Play Services is an add on to android to allow app developers easily interface with backend services Google provides.
- Distributed with the Play Store and updated silently in the background.
- Not on devices without the Play Store.

What does it provide

- Security services. App scanning and ability to update security provider, etc.
- Maps - Ability to incorporate Google Maps into your app.
- Location - Fused location provider, Geofences, etc.
- Google Cloud Messaging (client)

What does it provide

- Places - look up places, cities, etc
- Android Wear - Watch communications
- Google Wallet - purchase physical goods
- Mobile Ads - Make money in your app
- Analytics - measure user actions in app
- Many more...

How to use

- First add the full library or subset of child libraries to your project.
- Activate the service you need on the Google Developer Console if needed.
- Add api key to your app if needed.

Restrictions on use

- Open source licenses. As a rule you should display the open source licenses you use in your app. This includes the support libraries.
  - You may see something buried deep in settings or help called “legal” or “licenses”, etc. This is what is used for.
  - Essentially a text view of all the licenses you use. You can make it fairly hard-coded, etc. Just needs to be there.

Restrictions on use cont.

- Google Places, Maps, etc may require you to display a logo of some sort. Exactly what the requirements are varies by api. Some such as places may require a logo depending on how the data was used.
  - This is true of most services. Yahoo, Bing etc. all require the same attribution.
  - Cost of a free service.

Error handling

- When connecting to Google play services, there is a bit of error handling in case the user “uninstalled” the service, or somehow didn’t get the update.
- For the most part it is straight forward to add to the app, but can be tricky. Luckily they keep making it easier.

Error handling cont.

- If you use many services throughout the app or have several fragments using services, you should check for the services in the launcher Activity.
- However, if you use only if a deeper activity/fragment, you may want to defer until the user gets there.

Basic Code Flow

- OnCreate() - initialize your api connection and establish your callback listener for the fragment or activity.
- OnStart() or onResume() - establish a connection
- OnStop() or onPause() disconnect
  - Be sure to use start/stop or resume/pause. Using the appropriate lifecycle pair keeps things happy.

The callbacks

- onConnected()
  - Your api is ready for use. Its a good place to start any asynchronous operations that do not require user interaction, or start loading view data (aka maps) that need a connection.
- onSuspended()
  - You lost connectivity. If you have background ops running that may depend on the connection, you should stop them here

The callbacks continued

- onConnectionFailed()
  - This is your opportunity to ensure play services is up to date, etc. If you are using this via the connection api, I'd just follow the code sample Google provides except using the new DialogFragment instead of rolling your own. (This was required not that long ago).

Places

We made it

Enough boilerplate, the good stuff

- Places API
  - Handy way to allow users to quickly find stores, parks, sites, cities, etc.
- Two main apis:
  - Place picker - A ui where the user picks a nearby location.
  - Autocomplete - Ability to search for places based on text user provides.

Autocomplete API

- Need three pieces of info:
  - The search term from the user
  - A Lat/Long rectangle used to restrict/prioritize the search. In the sample it is restricted to the city of Seattle. If the app was location based, I'd compute a boundary to search in.
  - (optional) A filter that restricts the autocomplete to certain types of information. The sample is restricting to "establishments".

Invoking the api

- The api returns a PendingResult which is essentially a handle to the query.
- The result lets to proceed in two ways:
  - Synchronously, wait for the result (good if in background thread).
  - Asynchronously, provide a callback that will be invoked with the result is ready.

Processing the result

- The places api gives you a “buffer” to the result. Essentially a temp memory you need to “release” when done.
- Best to copy the data out of the buffer, but optionally you could “freeze” an object which does something similar.

Restrictions on data storage

- In the sample you will notice I do not persist the data to a content provider.
- Only allowed to temporarily cache for use in same UI context for up to 30 days with mandatory flushing.
  - In context of this app, I could cache the results and then query them and “intermix” with live data.
- However, the placeId may be stored indefinitely to allow looking up data for a place again or fetching more details.

Play Services and Places Demo

Bitmap loading

Bitmap loading

- Conceptually a simple problem, but many ways to go wrong.
- High level ops:
  - Establish HttpUrlConnection
  - If status successful, convert an input stream to a bitmap.

Where things go wrong

- Recycling views
  - Easiest place to go wrong is not handling view recycling. Aka image view in a list or grid view.
  - Essentially start loading in view 1. Bitmap/network slow so it takes enough time the user scrolls down the list before it loads.
  - View 1 recycled for use by view 9.
  - View 9 start image load and is cached so shown.
  - View 1 finishes loading and replaces image in view 9

Where things go wrong cont.

- User sees wrong image.
  - Impact may be user doesn’t realize its wrong.
  - User sees image for “Game of Thrones” show in description talking about “Modern Family”. User questions app reliability.

Other problems

- Image heavy app blocks loading for other content, if images not restricted to limited resources.
- Caching - Images use most memory, but also tend to be most expensive to load even from disk.

Other problems cont.

- Need to resize. Server providing images too large for space. For example, no thumbnail version.
  - Requires app to post process image after downloading, but before showing. Ideally the smaller copy is saved to avoid doing on every load.

Image Reuse

- Bitmaps consume a lot of memory but with API 11 (Honeycomb), you can reuse that memory.
  - Avoid thrashing where GC cleans up bitmap as allocating new ones.
  - However, prior to API 19, the bitmaps had to be same size.
  - Most 3rd parties do not take advantage of this.

Your solution

- Use a third party library that handles these issues for you (or most of them)
  - Glide
  - Picasso
  - Volley (Part of AOSP)
- For HW2 you should use Glide/Picasso as it is easier to setup. Volley will work as well, but requires a manual library project.

Bitmap recycling demo

Resources

Resources

- Setup Google Play Services
- Accessing google play services
- Google Places Android
- Glide
- Picasso
- Volley