Introduction to Mobile Programming

Android Programming

Chapter 6

Location-based Services

. Battery Optimization

1. Background location gathering is throttled and location is computed, and delivered only a few times an hour.

2. Wi-Fi scans are more conservative, and location updates aren't computed when the device stays connected to the same static access point.

3. Geofencing responsiveness changes from tens of seconds to approximately two minutes. This change noticeably improves battery performance—up to 10 times better on some devices.

Battery Drain

1. ACCURACY

   The precision of the location data. In general, the higher the accuracy, the higher the battery drain.

2. FREQUENCY

   How often location is computed. The more frequent location is computed, the more battery is used.

3. LATENCY

   How quickly location data is delivered. Less latency usually requires more battery.

ACCURACY

- setPriority()

  - `PRIORITY_HIGH_ACCURACY` provides the most accurate location possible, which is computed using as many inputs as necessary (it enables GPS, Wi-Fi, and cell, and uses a variety of Sensors), and may cause significant battery drain.
  - `PRIORITY_BALANCED_POWER_ACCURACY` provides accurate location while optimizing for power. Very rarely uses GPS. Typically uses a combination of Wi-Fi and cell information to compute device location.
  - `PRIORITY_LOW_POWER` largely relies on cell towers and avoids GPS and Wi-Fi inputs, providing coarse (city-level) accuracy with minimal battery drain.
  - `PRIORITY_NO_POWER` receives locations passively from other apps for which location has already been computed.

FREQUENCY

- Use the `setinterval()` method to specify the interval at which *location is computed for your app*.
- Use the `setFastestInterval()` to specify the interval at which *location computed for other apps is delivered to your app*.

LATENCY

1. setMaxWaitTime()

If your app doesn't immediately need a location update, you should pass the largest possible value to the `setMaxWaitTime()` method, effectively trading latency for more data and battery efficiency.

When using geofences, apps should pass a large value into the `setNotificationResponsiveness()` method to preserve power. A value of five minutes or larger is recommended.

Location Use Cases

User visible or foreground updates

A mapping app that needs frequent, accurate updates with very low latency. All updates happen in the foreground: the user starts an activity, consumes location data, and then stops the activity after a short time.

Use the `setPriority()` method with a value of `PRIORITY_HIGH_ACCURACY` or `PRIORITY_BALANCED_POWER_ACCURACY`.

Knowing the location of the device

A weather app wants to know the device's location.

Use the `getLastLocation()` method, which returns the most recently available location (which in rare cases may be null) . This method provides a simple way of getting location and doesn't incur costs associated with actively requesting location updates. Use in conjunction with the `isLocationAvailable()` method, which returns `true` when the location returned by `getLastLocation()` is reasonably up-to-date.

Starting updates when a user is at a specific location

Requesting updates when a user is within a certain distance of work, home, or another location.

Use geofencing in conjunction with fused location provider updates. Request updates when the app receives a geofence entrance trigger, and remove updates when the app receives a geofence exit trigger. This ensures that the app gets more granular location updates only when the user has entered a defined area.

Location Use Cases

Starting updates based on the user's activity state

Requesting updates only when the user is driving or riding a bike.

Use the Activity Recognition API in conjunction with fused location provider updates. Request updates when the targeted activity is detected, and remove updates when the user stops performing that activity.

Long running background location updates tied to geographical areas

The user wants to be notified when the device is within proximity of a retailer.

This is an excellent use case for geofencing. Because the use case almost certainly involves background location, use the `addGeofences(GeofencingRequest, PendingIntent)` method.

Long running background location updates without a visible app component.

An app that passively tracks location

Use the `setPriority()` method with the `PRIORITY_NO_POWER` option if possible because it incurs almost no battery drain. If using `PRIORITY_NO_POWER` isn't possible, use `PRIORITY_BALANCED_POWER_ACCURACY` or `PRIORITY_LOW_POWER`, but avoid using `PRIORITY_HIGH_ACCURACY` for sustained background work because this option substantially drains battery.

Location Best Practices

Remove location updates `requestlocationUpdates()` `removeLocationUpdates()`

Set timeouts `setExpirationDuration()`, `setExpirationTime()`

Batch Requests

```
LocationRequest request = new LocationRequest();
request.setInterval(10 * 60 * 1000);
request.setMaxWaitTime(60 * 60 * 1000);
```

Use passive location updates

```
LocationRequest request = new LocationRequest();
request.setInterval(15 * 60 * 1000);
request.setFastestInterval(2 * 60 * 1000);
```

Get the Last Known Location

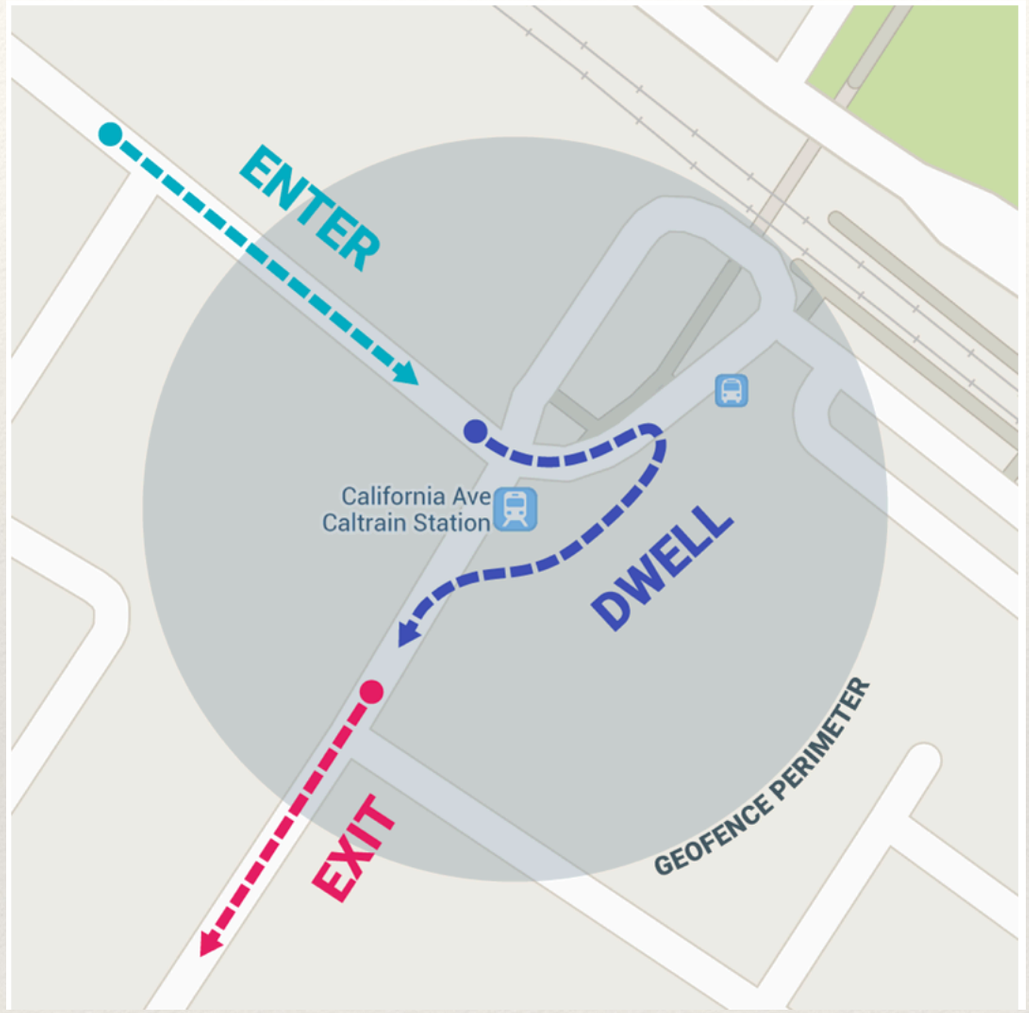
- Setup Google Play Services
- Specify app permissions
- Create Location Services client
- Get the Last Known Location

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.android.gms.location.sample.basiclocationsample" >

  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
</manifest>
```

```
private FusedLocationProviderClient fusedLocationClient;

// ..

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...

    fusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
}
```

```
fusedLocationClient.getLastLocation()
        .addOnSuccessListener(this, new OnSuccessListener<Location>() {
            @Override
            public void onSuccess(Location location) {
                // Got last known location. In some rare situations this can be null.
                if (location != null) {
                    // Logic to handle location object
                }
            }
        });
```

Create and Monitor Geofences



Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

Set up for Geofence Monitoring

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

<!-- Required if your app targets Android 10 (API level 29) or higher -->
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
```

```
private GeofencingClient geofencingClient;

@Override
public void onCreate(Bundle savedInstanceState) {
    // ...
    geofencingClient = LocationServices.getGeofencingClient(this);
}
```

Create and Add GeoFences

```
geofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.getKey())

    .setCircularRegion(
            entry.getValue().latitude,
            Constants.GEOFENCE_RADIUS_IN_METERS
    )
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
            Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```

Specify Geofences and Initial Triggers

```
private GeofencingRequest getGeofencingRequest() {
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);
    builder.addGeofences(geofenceList);
    return builder.build();
}
```

Define a Broadcast Receiver for Geofence Transitions

```
public class MainActivity extends AppCompatActivity {

    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (geofencePendingIntent != null) {
            return geofencePendingIntent;
        }
        Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when
        // calling addGeofences() and removeGeofences().
        geofencePendingIntent = PendingIntent.getBroadcast(this, 0, intent, PendingInter
                FLAG_UPDATE_CURRENT);
        return geofencePendingIntent;
    }
```

Add Geofences

```
geofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
        .addOnSuccessListener(this, new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                // Geofences added
                // ...
            }
        })
        .addOnFailureListener(this, new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                // Failed to add geofences
                // ...
            }
        });
```

Handle Geofence Transitions

```
public class GeofenceBroadcastReceiver extends BroadcastReceiver {
    // ...
    protected void onReceive(Context context, Intent intent) {
        GeofencingEvent geofencingEvent = GeofencingEvent.fromIntent(intent);
        if (geofencingEvent.hasError()) {
            String errorMessage = GeofenceStatusCodes.getErrorString(geofencingEvent.getErrorCode()
            Log.e(TAG, errorMessage);
            return;
        }

        // Get the transition type.
        int geofenceTransition = geofencingEvent.getGeofenceTransition();

        // Test that the reported transition was of interest.
        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER ||
                geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

            // Get the geofences that were triggered. A single event can trigger
            // multiple geofences.
            List<Geofence> triggeringGeofences = geofencingEvent.getTriggeringGeofences();

            // Get the transition details as a String.
            String geofenceTransitionDetails = getGeofenceTransitionDetails(
                    this,
                    geofenceTransition,
                    triggeringGeofences
            );

            // Send notification and log the transition details.
            sendNotification(geofenceTransitionDetails);
            Log.i(TAG, geofenceTransitionDetails);
        } else {
            // Log the error.
            Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
                    geofenceTransition));
        }
    }
}
```

Stop Geofence Monitoring

```
geofencingClient.removeGeofences(getGeofencePendingIntent())
        .addOnSuccessListener(this, new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                // Geofences removed
                // ...
            }
        })
        .addOnFailureListener(this, new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                // Failed to remove geofences
                // ...
            }
        });
```

Location Object may be NULL

- Location is turned off in the device settings. The result could be `null` even if the last location was previously retrieved because disabling location also clears the cache.
- The device never recorded its location, which could be the case of a new device or a device that has been restored to factory settings.
- Google Play services on the device has restarted, and there is no active Fused Location Provider client that has requested location after the services restarted. To avoid this situation you can create a new client and request location updates yourself. For more information, see Receiving Location Updates.

Change Location Settings

- Set up a Location Request
- Update Interval
- Fastest Update Interval
- Priority

```
protected void createLocationRequest() {
    LocationRequest locationRequest = LocationRequest.create();
    locationRequest.setInterval(10000);
    locationRequest.setFastestInterval(5000);
    locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
}
```

```
LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder()
     .addLocationRequest(locationRequest);
```

Request Location Updates

```
@Override
protected void onResume() {
    super.onResume();
    if (requestingLocationUpdates) {
        startLocationUpdates();
    }
}

private void startLocationUpdates() {
    fusedLocationClient.requestLocationUpdates(locationRequest,
            locationCallback,
            null /* Looper */);
}
```

```
private LocationCallback locationCallback;

// ...

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...

    locationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(LocationResult locationResult) {
            if (locationResult == null) {
                return;
            }
            for (Location location : locationResult.getLocations()) {
                // Update UI with location data
                // ...
            }
        };
    };
}
```

Stop Location Updates

```
@Override
protected void onPause() {
    super.onPause();
    stopLocationUpdates();
}

private void stopLocationUpdates() {
    fusedLocationClient.removeLocationUpdates(locationCallback);
}
```

```
@Override
protected void onResume() {
    super.onResume();
    if (requestingLocationUpdates) {
        startLocationUpdates();
    }
}
```