Xử lý bất đồng bộ

MultiUni
Trần Vũ Tất Bình

Thread

- Thông thường dùng Thread để xử lý các code xử lý nặng, hoặc mất thời gian có thể gây chậm chương trình hoặc giao diện bị block.
- Thread khá thông dụng và trong Android dùng lớp Thread của Java.

http://developer.android.com/reference/java/lang/Thread.html

- Mặc định, mỗi ứng dụng chạy trong một process và code được thực thi trong thread chính của process đó.

Thread

- Nếu code xử lý quá lâu, không kịp phản hồi lại các sự kiện người dùng trong 5 giây thì sẽ xuất hiện dialog “Application is not responding” và người dùng có thể force close ứng dụng ngay lập tức.
- Dù không bị force close thì việc ứng dụng bị lag là khó chấp nhận.
- Tham khảo 3 link sau:

http://developer.android.com/guide/practices/design/responsiveness.html
http://developer.android.com/guide/practices/design/seamlessness.html
http://developer.android.com/guide/practices/design/performance.html

Thread

```
Thread thread = new Thread() {
    @Override
    public synchronized void start() {
        // Khởi tạo các đối tượng cần thiết tại đây
        super.start();
    }

    @Override
    public void run() {
        // code xử lý chính của thread trong này
        super.run();
    }
};
thread.start(); //bắt đầu thread
```

Thread

- Lưu ý:
  - Thread lần đầu thực thi gọi phương thức start(), những lần sau chỉ gọi phương thức run(), không gọi start() nữa.
  - Các code xử lý liên quan đến giao diện chỉ được xử lý trong thread chính của ứng dụng (ví dụ load ảnh từ mạng về thì dung thread, nhưng hiển thị ảnh lên ImageView thì xử lý trong thread chính)
  - Sau khi thực thi xong phương thức run(), thread không còn active nữa.

Handler

- Trong Android, để tiện việc giao tiếp giữa 2 thread (như đề cập ở slide trước) ta dùng đối tượng Handler.
- Ngoài ra, có thể dùng Handler để đặt xử lý một yêu cầu nào đó sau một khoảng thời gian xác định.
- Chi tiết tại đây:

  http://developer.android.com/reference/android/os/Handler.html

Handler

- Giao tiếp giữa 2 Thread:
  - Giả sử trong phương thức run() của Thread trong slide trước, đã lấy xong đối tượng Bitmap về. Muốn truyền đối tượng Bitmap cho Thread chính hiển thị lên màn hình:

```
Message msg = mHandler.obtainMessage(1, bitmap);
mHandler.sendMessage(msg);
```

  - Trong code của Activity (mặc định là thread chính), ta khai báo một đối tượng Handler tương tứng như sau:

Handler

```
Handler mHandler = new Handler() {

    @Override
    public void handleMessage(Message msg) {
        if (msg.what == 1) {
            //Hiển thị Bitmap
            mImageView.setImageBitmap((Bitmap)msg.obj);
        }
        super.handleMessage(msg);
    }
};
```

Handler

- Vừa rồi, ta nhờ đối tượng mHandler lấy ra một message và gắn mã vào cho message đó, kèm theo đối tượng bitmap. Sau đó gửi đi.
- Message gửi đi sẽ được nhận phương thức callback là handleMessage() của đối tượng Handler.
- Handler còn có thể gửi message để xử lý sau một khoảng thời gian định sẵn sendMessageAtTime hoặc xử lý tại một thời điểm định sẵn sendMessageDelayed … có thể tìm hiểu thêm trong tài liệu của lớp Handler

Handler

- Handler được tạo trong thread nào thì sẽ sử dụng message queue của thread đó.
- Có thể dùng Handler như bộ đếm giây khi chơi nhạc, hoặc chức năng tương tự ☺
- Lưu ý là nếu trong message queue vẫn còn message thì vẫn còn thực thi dù đã thoát khỏi ứng dụng, cẩn thận chỗ này chứ không ứng dụng chạy bậy đó.

AlarmManager

- Dùng AlarmManager để thực hiện đăng ký xử lý một thao tác nào đó tại một thời điểm nhất định trong tương lai (thường là thời gian dài).
- Nếu xử lý trong thời gian ngắn thì khuyến cáo nên dùng Handler.
- Ưu điểm của AlarmManager, khi đến thời điểm được đựt trước, dù ứng dụng đang không chạy vẫn được gọi.
- Nếu tắt máy thì bật lại cũng không còn (lưu ý điểm này)

AlarmManager

- Khởi tạo một alarm:

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

Intent broadcastIntent = new Intent("org.multiuni.android.ACTION...");

PendingIntent pendingIntent = PendingIntent.getBroadcast(this,
        0, broadcastIntent, PendingIntent.FLAG_CANCEL_CURRENT);

am.set(AlarmManager.RTC_WAKEUP, triggerAtTime, pendingIntent);
```

AlarmManager

- Giải thích:
  - Khởi tạo một đối tương AlarmManager để làm việc với Alarm.
  - Tạo một intent tên broadcastIntent, intent này được dùng để gửi broadcast khi đến thời điểm định sẵn.
  - PendingIntent được khởi tạo gồm context, broadcastIntent ở trên và một cờ báo rằng nếu đã có một Alarm tương tự thì bỏ nó đi và dùng cái mới này.

- Sau cùng, set alarm với 3 thông số:
  - Bộ đếm thời gian (có 4 loại, xem trong document của AlarmManager)
  - Thời gian chính xác để bật alarm lên.
  - PendingIntent gửi đi (dùng để xác định tới thời điểm bật alarm lên thì cần phát intent nào)

- Tham khảo thêm tại đây:

http://developer.android.com/reference/android/app/AlarmManager.html

Notification

- Trong những trường hợp các bạn muốn hiện một thông báo về một sự kiện nào đó cho người dùng mà khong muốn ảnh hưởng đến công việc của họ hoặc không chắc họ có đang cầm điện thoại (tin nhắn, cuộc gọi, email…)
- Hoặc bạn muốn hiển thị thông tin một việc nào đó đang xảy ra trên điện thoại và mong người dùng biết (đang nghe nhạc, đang trong cuộc gọi, thiếu thẻ nhớ…)

→ Notification

Notification

- Bạn có thể tạo một notification có âm báo, rung, đèn led, icon…
- Notification có 2 dạng:
  - One time
  - On going
- Xem chi tiết tại

http://developer.android.com/reference/android/app/NotificationManager.html

Notification

- Code demo chi tiết:

ApiDemos → com.example.android.apis.app → StatusBarNotification.java