理解 RxJava 的线程模型（下）

2016-08-02 安卓应用频道

(点击上方公众号，可快速关注)

> 来源：鸟窝（@colobu ）
> 链接：colobu.com/2016/07/25/understanding-rxjava-thread-model/

接上文

五、Schedulers

上面我们了解了RxJava可以使用subscribeOn和observeOn可以改变和切换线程，以及onNext是顺序执行的，不是并发执行,至多也就切换到另外一个线程，如果它中间的操作是阻塞的，久会影响整个Rx的执行。

Rx是通过调度器来选择哪个线程执行的，RxJava内置了几种调度器，分别为不同的case提供线程：

- **io()**：这个调度器时用于I/O操作, 它可以增长或缩减来确定线程池的大小它是使用CachedThreadScheduler来实现的。需要注意的是，它的线程池是无限制的，如果你使用了大量的线程的话，可能会导致OutOfMemory等资源用尽的异常。
- **computation()**：这个是计算工作默认的调度器，它与I/O操作无关。它也是许多RxJava方法的默认调度器：buffer(),debounce(),delay(),interval(),sample(),skip()。

因为这些方法内部已经调用的调度器，所以你再调用subscribeOn是无效的，比如下面的例子总是使用computation调度器的线程。

```
Observable.just(1,2,3)
        .delay(1, TimeUnit.SECONDS)
        .subscribeOn(Schedulers.newThread())
        .map(i -> {
            System.out.println("map: " + Thread.currentThread().getName());
            return i;
        })
```

```
        .subscribe(i -> {});
```

- **immediate()** :这个调度器允许你立即在当前线程执行你指定的工作。它是timeout(),timeInterval(),以及timestamp()方法默认的调度器。
- **newThread()** :创建一个新的线程只从。
- **trampoline()** :为当前线程建立一个队列，将当前任务加入到队列中依次执行。

同时，Schedulers还提供了from静态方法，用户可以定制线程池：

```
ExecutorService es = Executors.newFixedThreadPool(200, new
ThreadFactoryBuilder().setNameFormat("SubscribeOn-%d").build());
Schedulers.from(es)
```

六、改造，异步执行

现在，我们已经了解了RxJava的线程运行，以及相关的调度器。可以看到上面的例子还是顺序阻塞执行的，即使是切换到另外的线程上，依然是顺序阻塞执行，显示它的吞吐率非常非常的低。下一步我们就要改造这个例子，让它能异步的执行。

下面是一种改造方案，我先把代码贴出来，再解释：

```
public static void testRxJavaWithFlatMap(int count) throws Exception {
    ExecutorService es = Executors.newFixedThreadPool(200, new
ThreadFactoryBuilder().setNameFormat("SubscribeOn-%d").build());
    URL url = new URL("http://127.0.0.1:8999/");
    CountDownLatch finishedLatch = new CountDownLatch(1);
    long t = System.nanoTime();
    Observable.range(0, count).subscribeOn(Schedulers.io()).flatMap(i -> {
            //System.out.println("A: " + Thread.currentThread().getName());
            return Observable.just(i).subscribeOn(Schedulers.from(es)).map(v -> {
                    //System.out.println("B: " + Thread.currentThread().getName());
                    try {
                        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
                        conn.setRequestMethod("GET");
                        int responseCode = conn.getResponseCode();
                        BufferedReader in = new BufferedReader(new
```

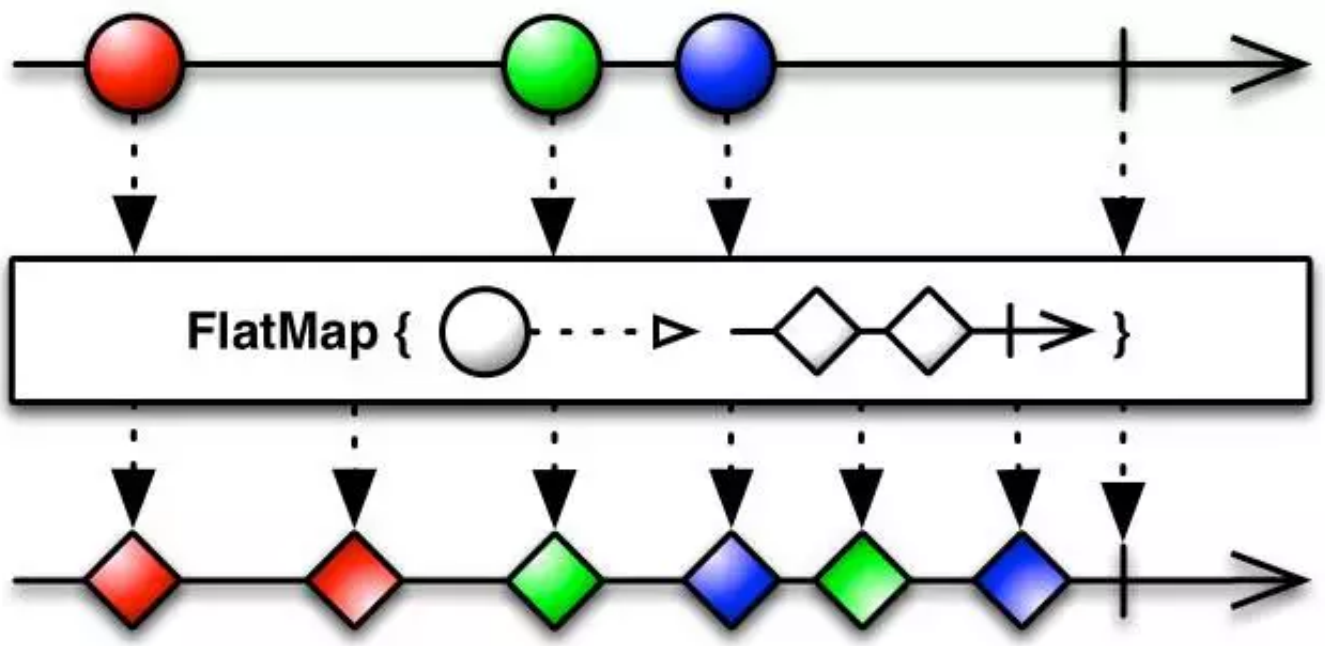
```
InputStreamReader(conn.getInputStream()));
                        String inputLine;
                        while ((inputLine = in.readLine()) != null) {
                            //response.append(inputLine);
                        }
                        in.close();
                        return responseCode;
                    } catch (Exception ex) {
                        return -1;
                    }
                }
            );
        }
    ).observeOn(Schedulers.computation()).subscribe(statusCode -> {
        //System.out.println("C: " + Thread.currentThread().getName());
    }, error -> {
    }, () -> {
        finishedLatch.countDown();
    });
    finishedLatch.await();
    t = (System.nanoTime() - t) / 1000000; //ms
    System.out.println("RxJavaWithFlatMap TPS: " + count * 1000 / t);
    es.shutdownNow();
}
```

通过flatmap可以将源Observable的元素项转成n个Observable,生成的每个Observable可以使用线程池并发的执行，同时flatmap还会将这n个Observable merge成一个Observable。你可以将其中的注释打开，看看线程的执行情况。

性能还不错：

RxJavaWithFlatMap TPS: **3906**。

> **FlatMap** — transform the items emitted by an Observable into Observables, then flatten the emissions from those into a single Observable



七、另一种解决方案

我们已经清楚了要并行执行提高吞吐率的解决办法就是创建多个Observable并且并发执行。基于这种解决方案，我们还可以有其它的解决方案。

上一方案中利用flatmap创建多个Observable,针对我们的例子，我们何不直接创建多个Observable呢？

```
public static void testRxJavaWithParallel(int count) throws Exception {
    ExecutorService es = Executors.newFixedThreadPool(200, new
ThreadFactoryBuilder().setNameFormat("SubscribeOn-%d").build());
    URL url = new URL("http://127.0.0.1:8999/");
    CountDownLatch finishedLatch = new CountDownLatch(count);
    long t = System.nanoTime();
    for (int k = 0; k < count; k++) {
        Observable.just(k).map(i -> {
            //System.out.println("A: " + Thread.currentThread().getName());
            try {
                HttpURLConnection conn = (HttpURLConnection) url.openConnection();
                conn.setRequestMethod("GET");
                int responseCode = conn.getResponseCode();
                BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
                String inputLine;
                while ((inputLine = in.readLine()) != null) {
```

```
                //response.append(inputLine);
            }
            in.close();
            return responseCode;
        } catch (Exception ex) {
            return -1;
        }
    }).subscribeOn(Schedulers.from(es)).observeOn(Schedulers.computation()).subscribe(statusCode -> {
    }, error -> {
    }, () -> {
        finishedLatch.countDown();
    });
}
finishedLatch.await();
t = (System.nanoTime() - t) / 1000000; //ms
System.out.println("RxJavaWithParallel TPS: " + count * 1000 / t);
es.shutdownNow();
}
```

性能更好一点：

RxJavaWithParallel2 TPS: **4716**。

这个例子没有使用Schedulers.io()作为它的调度器，这是因为如果在大并发的情况下，可能会出现创建过多的线程导致资源不错，所以我们限定使用200个线程。

八、总结

- **subscribeOn()** 改变的Observable运行(operate)使用的调度器，多次调用无效。
- **observeOn()** 改变Observable发送notifications的调度器，会影响后续的操作，可以多次调用
- 默认情况下， 操作链使用的线程是调用subscribe()的线程
- Schedulers提供了多个调度器，可以并行运行多个Observable
- 使用RxJava可以实现异步编程，但是依然要小心线程阻塞。而且由于这种异步的编程，调试代码可能更加的困难

九、参考文档

- http://reactivex.io/documentation/contract.html
- http://reactivex.io/documentation/operators/subscribeon.html 中文翻译
- http://reactivex.io/documentation/operators/observeon.html 中文翻译
- http://reactivex.io/documentation/scheduler.html
- http://tomstechnicalblog.blogspot.com/2016/02/rxjava-understanding-observeon-and.html
- http://tomstechnicalblog.blogspot.com/2015/11/rxjava-achieving-parallelization.html
- https://medium.com/@diolor/observe-in-the-correct-thread-1939bb9bb9d2 中文翻译
- https://github.com/mcxiaoke/RxDocs